Off-Line Performance Prediction of Message-Passing Applications on Cluster Systems*

E. Mancini[1], M. Rak[2], R. Torella[2], and U. Villano[3]

[1] RCOST, Università del Sannio, Via Traiano, 82100 Benevento, Italy
epmancini@unisannio.it
[2] DII, Seconda Università di Napoli, via Roma 29, 81031 Aversa (CE), Italy
massimiliano.rak@unina2.it, ganglio@kaiba.cc
[3] Dip. di Ingegneria, Università del Sannio, C.so Garibaldi 107, 82100 Benevento, Italy
villano@unisannio.it

Abstract. This paper describes a simulation-based technique for the performance prediction of message-passing applications on cluster systems. Given data measuring the performance of a target cluster in the form of standard benchmark results, along with the details of the chosen computing configuration (e.g., the number of nodes), it is possible to build and to validate automatically a detailed simulation model. This makes it possible to predict the performance of fully-developed or skeletal code off-line, i.e., without resorting to the real hardware. The reasonable accuracy obtained makes this approach particularly useful for preliminary performance testing of parallel code on non-available hardware. After a description of the approach and of the construction and validation of the simulation model, the paper presents a case study.

1 Introduction

The use of cluster architectures for solving computing-intensive problems is currently customary both in the industrial research and academic communities. The impressive computing power of even small-sized and self-made clusters has changed the way most people regards these architectures. As a matter of fact, clusters are no longer considered as the poor man's supercomputer, and currently more than 90 clusters are present in the list of the top 500 supercomputer sites [1].

The awareness of the high computing power and of the wide range of application software available for such systems has made the use of cluster architectures attractive, widening the number of possible users. The result is that now clusters are precious (and expensive) resources, potentially useful for a very high number of users. On the bad side, the often-conflicting computing needs of these users have to be suitably managed, thus calling for the adoption of effective job management systems. State-of-the-art job management systems (e.g., the Portable Batch System [2]) make it possible to enforce site-specific scheduling policies for running user jobs in both time-shared and space-shared modes. Many sites, besides a batching system, also

* This work was partially supported by Regione Campania, project l. 41/2000 "Utilizzo di predizioni del comportamento di sistemi distribuiti per migliorare le prestazioni di applicazioni client/server basate su web"

adopt an advanced scheduler (e.g., Maui [3]) providing mechanisms to optimize the use of computing resources, to monitor system performance, and, more generally, to manage effectively the system.

As a matter of fact, current cluster job management systems are not able to satisfy completely cluster user management needs, and the development of advanced schedulers and batching system will be a strategic research field in the near future. However, it is clear that a more efficient use of existing systems could be attained if:

- as many users as possible are diverted from the main system, encouraging them to use off-line facilities for software development and performance testing;
- for each submitted job, reasonable predictions of its impact on system workload (number of computing nodes used, running time, use of system resources, ...) are available. This would allow more effective scheduling decisions to be made.

In light of all the above, the adoption of simulation-based performance prediction techniques appears a particularly promising approach. A most obvious objective is to obtain off-line (i.e., without accessing the "real" cluster hardware) reasonably accurate predictions of software performance and resource usage to be used to guide scheduling choices. Less obvious, but probably of much greater importance, is the possibility to compare several different hardware configurations, in order to find one compatible with the expected performance. For example, preliminary simulation-based performance analysis could suggest the use of larger or of a differently configured system [4]. But the use of simulation-based performance analysis environments has also great potential for performance-driven parallel software development. This process uses quantitative methods to identify among the possible development choices the designs that are more likely to be satisfactory in terms of performance [5]. The final result is not only a reduction of the time and effort invested in design (and coding), but also less cluster workload, due to reduced testing and benchmarking times.

This paper proposes a complete development and performance analysis process for the performance prediction of message-passing applications on cluster systems. The method relies on the use of an existing and fairly mature simulation environment (HeSSE, [6,7,8]). Given data measuring the performance of a target cluster in the form of standard benchmark results, along with trivial details of the chosen computing configuration (e.g., the number of nodes), it is possible to build a detailed simulation model by means of a user-friendly graphical interface. A recently-developed software makes then possible to tune and to validate automatically the model. After that, the user can predict the performance of fully-developed or even skeletal code and to obtain information on program behavior (performance summaries, animations, activity traces,...) in a simulated environment, without resorting to the real hardware.

The reasonable accuracy obtained (errors are typically below 10%) makes this *off-line* approach particularly useful for preliminary performance testing of parallel code, during software development steps, or simply when access to the real cluster hardware is not possible or is uneconomical. A most important point to be stressed here is that the cluster performance is measured simply by running a customary benchmark suite. In other words, the benchmark outputs are not used to get qualitative information on cluster performance (as is done customarily), but to build a simulation model that can produce quantitative information on the hardware/software behavior. This approach has never been followed in our previous work, and, to the best of our knowledge, is not even present in the literature.

This paper is structured as follows. The next section will introduce the HeSSE simulation environment and its modeling methodology. Section 3 describes the proposed modeling process, describing the automatic validation system. Section 4 shows a case study, where the proposed process is applied to the modeling of a simple application on two different cluster systems. After an examination of related work, the paper closes with our conclusions and the directions of future research.

2 HeSSE (Heterogeneous System Simulation Environment)

HeSSE (*He*terogeneous *S*ystem *S*imulation *E*nvironment) is a simulation tool that can reproduce or predict the performance behavior of a (possibly heterogeneous) distributed system for a given application, under different computing and network load conditions [6-10]. The distinctive feature of HeSSE is the adoption of a compositional modeling approach. Distributed heterogeneous systems (DHS) are modeled as a set of interconnected components; each simulation component reproduces the performance behavior of a section of the complete system at a given level of detail. Support components are also provided, to manage simulation-support features such as statistical validation and simulation restart. System modeling is thus performed primarily at the logical architecture level.

HeSSE is capable of obtaining simulation models for DHS systems that can be even very complex, thanks to the use of component composition. A HeSSE simulation component is basically a hard-coded object that reproduces the performance behavior of a specific section of a real system. More detailed, each component has to reproduce both the *functional* and the *temporal behavior* of the subsystem it represents. In HeSSE, the functional behavior of a component is the collection of the services that it exports to other components. In fact, components can be connected, in such a way that they can ask other components for services. On the other hand, the temporal behavior of a component describes the time spent servicing.

Applications are described in HeSSE through traces. A trace is a file that records all the relevant actions of the program. For example, a typical HeSSE trace file for parallel PVM applications is a timed sequence of CPU bursts and of requests to the run-time environment. Each trace represents a single execution of a parallel program. Traces can be obtained by application instrumentation and execution on a host system [9,10], or through prototype-oriented software description languages [6,11,12].

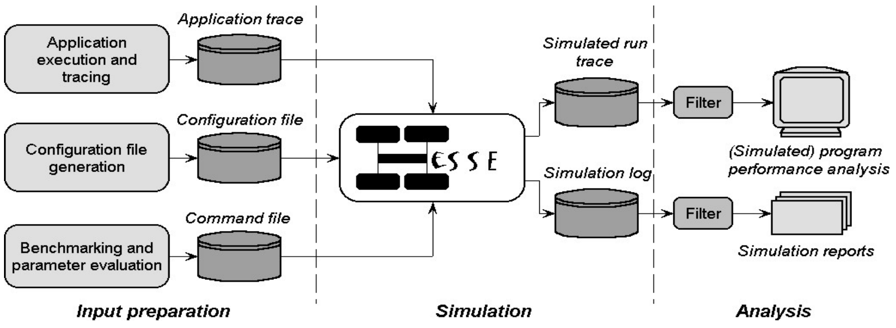

Fig. 1. HeSSE Performance Analysis Process

Parallel application behavioral analysis takes place as shown in Figure 1. The issue is further described in companion papers [4,6-12]. Previously published results show that the simulation environment is simple and effective to analyze real-world scientific applications with reasonable accuracy [4].

The stress in all the above-mentioned papers on the HeSSE simulation environment is on behavioral modeling of applications; the system modeling process is simply left to the performance analyzer expertise. In practice, a skilled analyzer can model system and application altogether, with relatively modest effort, and validate them as a whole. The procedure canonically followed requires getting the hardware parameters needed by the simulation environment through tests on the real system. These make it possible to reproduce correctly the application execution behavior. Successively, the tentative model is validated, by comparing its results with those obtained on the real system. It can be easily recognized that this simple approach fails whenever the target system is not readily available, thus precluding both the measurement of system parameters, needed for model construction, and its final validation. Off-line model construction and performance prediction is a challenging problem, and requires the use of new approaches.

The observation at the basis of the work described in this paper is that, even if the system is not available, it is almost always possible to obtain from cluster administrators the results of standard benchmarks, customarily used to evaluate the performance of their hardware. Benchmarks are commonly used to compare parallel and distributed systems, and they can be often considered as representative of the system "standard" use. As far as performance prediction and analysis is concerned, an obvious issue is to ascertain whether benchmarks can be used to tune and to validate a cluster simulation environment, or not. This is the problem tackled in the following section, where a modeling process based on benchmark results is proposed. The validity of the approach is considered in Section 4, which shows the accuracy results obtained for a simple problem on two different clusters.

3 The Proposed Modeling Process

Given a cluster system, the modeling and analysis process can be carried out in the HeSSE simulation environment in three macro steps ("macro" steps, since each step includes many different operations), which are briefly outlined in the following.

3.1 System High-Level Description

A very simple model of the system is obtained using natural language descriptions. This "base" model includes just information on the number of the computing nodes, the middleware adopted, and the network architecture.

High-level models can be built through the simple graphical interface provided by *HeSSEgraphs*. This is a visual tool that makes it possible to create interactively a visual model of a given DHS system. The HeSSEgraphs tool makes it also possible to obtain from the graphical description of the system the command and configuration files needed to simulate the system in the HeSSE environment. In HeSSEgraphs, a system is modeled as a plain graph, using predefined components collected in libraries. HeSSEgraphs shows all these components, grouped by library, in a left frame. The user can choose a component, and place it in a working drawing area by a drag-and-drop operation. The first step is the creation of nodes. Then he/she can add connections between them, and edit the properties of each node or connection using the mouse.

Figure 2 shows a screenshot of the visual tool HeSSEgraphs. The graph in the Figure is representative of a 4-node SMP system with Ethernet connectivity running a PVM application. Specialized off-the-shelf nodes model the computing node, the PVM daemons, every instance of the application, the NICs and a connecting Ethernet switch. It is of paramount importance to point out that the information managed at this level of abstraction can be easily retrieved from simple textual and rather informal descriptions of the cluster hardware. In practice, it is sufficient to know the number of nodes and their type (mono or multi-processor), and the network technology adopted to build the DHS graph using the nodes in the predefined libraries.

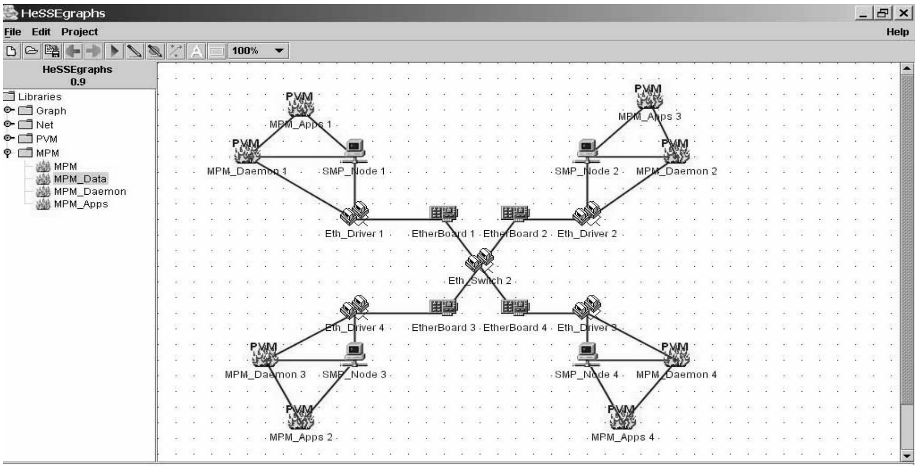

Fig. 2. HeSSEgraphs screenshot

3.2 Model Tuning and Benchmark Simulation

High-level descriptions can be built easily, thanks to the use of predefined components, but they cannot capture the behavior of real system, since they do not contain any type of temporal information. In fact, a high-level model does not describe a given cluster, but only a cluster class, to which the given target system belongs. The objective of the next modeling step, *Model Tuning*, is to define all the parameters that specialize the cluster class and allow to obtain a close description of the given computer system.

While high-level description was built by means of simple components, which can be gathered from a natural language description, parameters needed in the model tuning are not equally simple to be defined and understood. Furthermore, they should be gathered from the target cluster through dedicated experiments. As mentioned in the previous section, the canonical way to carry out this procedure is to rely on the analyzer experience and to perform intensive testing on the real hardware. The procedure followed here is based instead on the use of new HeSSE components that make it possible to perform the tuning in an automated way. The procedure is based on the use of previously collected cluster performance parameters. Instead of dedicated and proprietary tests, our choice was to adopt standard benchmarks for both parameters evaluation and simulation model validation.

The process takes place in the following steps:

- standard benchmark results are collected on the target system;
- a special HeSSE configuration file is built, adding tuning components to the high-level model previously defined;
- an automated learning process takes place;
- the simulation error is evaluated, validating the model.

3.2.1 Cluster Benchmarking

The Benchmark suite adopted for all the experiments described below is the Parkbench suite [13]. This is a well-known set of benchmark tests, freely available and largely used in the parallel computing community. With all the caveats linked to the use of standard benchmarks for cluster performance evaluation, the proposed tests are commonly considered very expressive of the overall system performance. In our experiments, we did not actually run the whole suite of benchmarks but only the MPBench group, aiming to tune the cluster configuration for MPI applications.

3.2.2 Tuner Configuration and Learning Process

The automation of the parameter fitting process relies on dedicated HeSSE components, able to manage a learning algorithm and benchmark reproduction. Usually benchmark programs contain loops whose body includes the tested operation sequence (along with timestamping instructions). A *Benchmark* component is able to reproduce the majority of available benchmarks, reading a benchmark description from a simple configuration file. A simple language was developed to describe this kind of benchmark tests, i.e., to provide information such as type and number of primitives to be tested (e.g. MPISend, MPIBroadcast, ...), and the number of test runs.

A *Trainer* component, instead, starts the simulation, asks the benchmark component to perform all the available tests, checks the time, compares the results with the reference ones (i.e., with the results of actual benchmark execution) and restarts the simulation with new parameters. The learning process stops when the accuracy is over a desired threshold. The learning procedure currently is based on simulated annealing fitting. The description of the learning algorithm details (e.g., convergence properties and execution time) can be found in [6].

3.2.3 Accuracy Evaluation and Model Validation

The trainer component reproduces only one half of the benchmark results available. For example, if the benchmarks are parameterized on message packet dimension, e.g. from 16 bytes to 1 MB in steps of 16 bytes, the simulation model reproduces the same tests in steps of 32 bytes. The remainder of the values (i.e., the other half of the benchmark results) is used to evaluate the model accuracy. This makes it possible to avoid obtaining a model that fits well the supplied benchmark values, but does not capture the overall performance behavior.

3.2.4 Application Analysis

Following up the evaluation of the model accuracy, the application performance analysis takes place, following the typical HeSSE steps: application tracing on a host system, simulation execution, and analysis of the performance results. The reader is referred to previous papers for details [4,6-12].

4 An Example: Cygnus and Cossyra Case Studies

The proposed approach was applied on two cluster systems of the PARSEC laboratory at the 2nd University of Naples, Cygnus and Cossyra. Though these systems are different as far as processors and node hardware are concerned, it will be shown that the accuracies obtained for their performance predictions are very similar.

Cygnus is a four SMP-node cluster. Each node is an SMP Compaq ProLiant 330 system, equipped with two Pentium IV, 512MB RAM memory. The network architecture is a Switched Fast Ethernet. The System is managed through Rocks [14], which currently includes Red Hat Linux 7.3 and well known administration tools, like PBS, MAUI and Ganglia. A Pentium III, 256 MB RAM, was adopted as frontend.

Cossyra is a 3 Alpha Workstation cluster. Each node is an AlphaStation 250, with Alpha RISC processor, 233 Mhz, 64MB RAM memory. Network is a standard 10 Mbs Ethernet. The operating system is Alpha Red Hat Linux 7.1.

4.1 Benchmark Results and Model Tuning

The benchmark-based model tuning was started by executing MPbench on both clusters. Then the automated tuning system was fed with the description of the benchmark set adopted and the benchmark results, thus starting the training phase. At the end of the learning step, all the simulation parameters found are collected in a report file. The resulting parameter values typically are a good fit to the training (benchmark) results. An example of the errors between the actual benchmark results and the tuned simulation model is proposed in Fig. 3. In this figure, the completion times of a simple *Roundtrip* benchmark are illustrated, as measured on the real cluster and obtained by simulation on the automatically-tuned system. Relative errors between measured data and simulation results, along with completion times, are reported in the same diagram in logarithmic scale and are always under 10%.

4.2 Target Application Performance Prediction

The application chosen to show the validity of the approach is a simple row-column square matrix multiplication. The algorithm adopted for parallel decomposition of the multiplication $C = A \times B$ works as follows: the master process splits the matrix A by rows in *numtask*-1 slices and sends one of them to each slave process. Then sends the matrix *B* to each slave. Slaves compute the row-column product between the slice of *A* and *B*, thus obtaining a slice of *C* that is sent back to the master. This simple algorithm [15] was chosen as a simple case study as the focus here is on the prediction methodology, rather than on parallel program development.

The application was instrumented and executed on a host system (the cluster frontend). The model was fed with the trace files thus obtained, and the application execution time was predicted for matrix sizes going from 10×10 to 500×500 in steps of 10. For each size, the tests on the real clusters were executed ten times. Each test, even for the largest matrices, required a few seconds of simulation. The actual and predicted performance results are shown in Figure 4. The proposed diagram shows (in logarithmic scale) both the real measurements on the cluster system and the simulation results, along with the resulting relative error. The target application simulation error has the same behavior shown in the benchmark execution. For both architectures, the resulting error is typically about 5%, and in any case under 10%. It is important to point out that the simulation results were obtained completely off-line, i.e., without resorting to the application execution on the clusters.

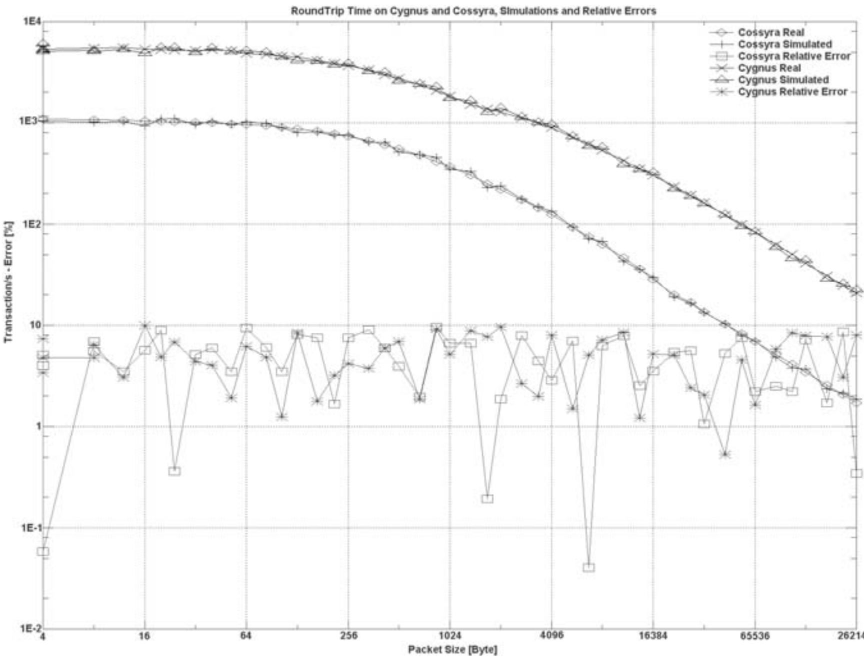

Fig. 3. Benchmark and simulation results after the learning phase, relative errors

5 Related Work

The problem of off-line cluster prediction has never been explicitly treated in existing literature. However, a wide number of papers deals with parallel and distributed system modeling and performance analysis [17-18], and it is likely that many existing simulation environments and tools could be applied to carry out performance analyses similar to those that are the object of this paper. Other authors focus their interest in tools for performance-driven software development, mainly in the context of Performance Engineering (see per example [19-20]).

Similarly, a wide body of literature deals with benchmarking and performance evaluation [21]. But benchmarking has never been used to characterize a system for performance simulation and to predict off-line the behavior of parallel applications.

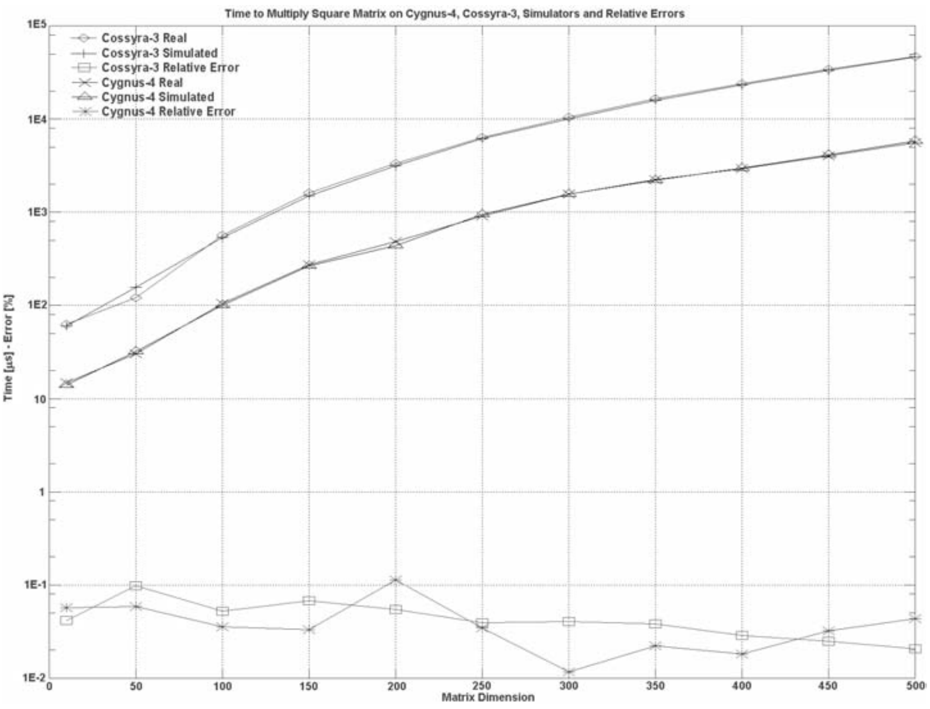

Fig. 4. Matrix multiplication actual and predicted performance results, relative errors

6 Conclusions and Future Work

This paper describes a modeling and performance prediction process that can be adopted even when the target system is not available. Standard benchmarks are used to tune and to validate the system simulation models, whose definition stems from simple natural language descriptions, and are obtained through a graphical interface. The accuracy results obtained (relative errors under 10%) make this approach attractive for off-line software development and analysis.

One of the main results of the work described here is the use of common benchmarks for tuning and validating the simulation models. The results obtained in our tests, some of which have been reported here, show that the execution of a benchmark is sufficient to fully characterize the target system for simulation.

A research field to be explored in the future is the possibility to compare the performance of proprietary applications on possibly unavailable target systems, using information gathered running standard benchmarks. This comparison can also be very useful when the clusters are components of large GRID systems. In this case, the objective can be to enable the user to choose the best system for his/her application execution, without expensive (in time and cost) test executions on the real hardware. A further evolution of the work documented in this paper is the prediction of benchmark results for non-existing clusters. We intend to study the possibility to predict the performance of applications by means only of artificial benchmark results.

References

1. TOP500 Supercomputer Sites, http://www.top500.org/
2. Portable Batch System, http://www.openpbs.org/main.html
3. Maui Scheduler, http://supercluster.org/maui/
4. Fahringer, T., Mazzocca, N., Rak, M., Villano, U., Pllana S., Madsen G.: Performance Modeling of Scientific Applications: Scalability Analysis of LAPW0. In Proc. PDP03 Conference, 3-7 February 2003, IEEE Press, Genova, Italy
5. Smith, C. U., Williams, L. G.: Performance Engineering Models of CORBA based Distributed-Object Systems. In: Computer Measurement Group Conf. Proceedings (1998)
6. Rak, M.: "A Performance Evaluation Environment for Distributed Heterogeneous Computer Systems Based on a Simulation Approach", Ph.D. Thesis, Facoltà di Ingegneria, Seconda Università di Napoli, Italy, 2002.
7. Mazzocca, N., Rak, M., Villano, U.: The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project. In: Dongarra, J. et al. (eds.): Recent Advances in Parallel Virtual Machine and Message Passing Interface. LNCS, Vol. 1908. Springer-Verlag, Berlin (2000) 266-273
8. Mazzocca, N., Rak, M., Villano, U.: Predictive Performance Analysis of Distributed Heterogeneous Systems with HeSSE. In Proc. 2001 Conference Italian Society for Computer Simulation (ISCSI01), CUEN, Naples, Italy (2002) 55-60
9. Aversa, R., Mazzeo, A., Mazzocca, N., Villano, U.: Heterogeneous System Performance Prediction and Analysis using PS. IEEE Concurrency, Vol. 6 (1998) 20-29
10. Aversa, R., Mazzeo, A., Mazzocca, N., Villano, U.: Developing Applications for Heterogeneous Computing Environments using Simulation: a Case Study. Parallel Computing, Vol. 24 (1998) 741-761
11. Mazzocca, N., Rak, M., Villano, U.: MetaPL: a Notation System for Parallel Program Description and Performance Analysis. In: Malyshkin, V. (ed.): Parallel Computing Technologies. LNCS, Vol. 2127. Springer-Verlag, Berlin (2001) 80-93
12. Mazzocca, N., Rak, M., Villano, U.: The MetaPL approach to the performance analysis of distributed software systems. In: Proc. 3rd International Workshop on Software and Performance (WOSP02), IEEE Press (2002) 142-149
13. PARKBENCH Report: Public International Benchmarks for Parallel Computers. Scientific Programming, Vol. 3 (1994) 101-146 (http://www.netlib.org/parkbench/)
14. Papadopoulos, P. M., Katz, M. J., Bruno, G.: NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters, In: Proc. IEEE Cluster 2001 (2001)
15. Center for Parallel Computer Training Courses, Matrix Multiplication Example http://www.pdc.kth.se/training/Tutor/MPI/Templates/mm/
16. Kerbyson, D. J., Papaefstatuiou, E., Harper, J. S., Perry, S. C., Nudd, G. R.: Is Predictive Tracing too late for HPC Users? In: Allan, M. F. Guest, D. S. Henty, D. Nicole and A. D. Simpson (eds.): Proc. HPCI'98 Conference: High Performance Computing. Plenum/Kluwer Publishing (1999) 57-67
17. Labarta, J., Girona, S., Pillet, V., Cortes T., Gregoris, L.: DiP: a Parallel Program Development Environment. Proc. Euro-Par '96, Lyon, France (Aug. 1996) Vol. II 665-674
18. Buck, J. T., et al.: A Framework for Simulating and Prototyping Heterogeneous Systems. Int. Journal of Comp. Simulation 4 (1994) 155-182
19. Smith C.U., Williams L.G.: Performance Engineering Evaluation of Object-Oriented Systems with SPEED. In: Marie, R. et. al. (eds.): Computer Perfomance Evaluation Modeling Techniques and Tools. LNCS, Vol. 1245, Springer-Verlag, Berlin (1997)
20. Menascé, D. A., Gomaa, H.: A Method for Design and Performance Modeling of Client/Server Systems. IEEE Trans. on Softw. Eng., Vol. 26 (2000) 1066-1085
21. Messina, P. et al.: Benchmarking advanced architecture computers, Concurrency: Practice and Experience, Vol. 2 (1990) 195-255