Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 363 – 370

ISSN 1896-7094

Performance Prediction Methods

Agnieszka Szymańska-Kwiecień[2], Jan Kwiatkowski[1], Marcin Pawlik[1],
Dariusz Konieczny[1]

[1] Institute of Applied Informatics
[2] Wroclaw Centre for Networking and Supercomputing
Wrocław University of Technology
Wybrzeze Wyspianskiego 27, 50-370 Wrocław Poland
{jan.kwiatkowski, marcin.pawlik, agnieszka.kwiecień,
dariusz.konieczny}@pwr.wroc.pl

Abstract. The increase of computer networks speed paired with the ubiquity of inexpensive, yet fast and generously equipped hardware offers many organizations an affordable way to increase the available processing power. Clusters, hyperclusters and even Grids, not so long ago seen only in huge datacenters, can now be found helping many small organizations in solving their computational needs. Finding an effective application performance prediction method constitutes one of particularly important but still open research problems. The paper presents the short overview of techniques utilized to predict application execution time. The whole spectrum of prediction methods is analyzed and selected tools dedicated for grid environments are presented.

1 Introduction

The increase of computer networks speed paired with the ubiquity of inexpensive, yet fast and generously equipped hardware offers many organizations an affordable way to increase the available processing power. Clusters, hyperclusters and even Grids, not so long ago seen only in huge datacenters, can now be found helping many small organizations in solving their computational needs. In many fields of science the increase of computational requirements made the access to powerful computational installations a necessity. The creation of a dedicated Beowulf type cluster, composed from cheap, commodity hardware is currently in the reach of many organizations. Scientists working for various projects, Clusterix—Polish National Grid project, The Enabling Grids for E-science (EGEE) project, GridLab project or Word Wide Grid project, to name only a few, are working on connecting resources distributed over local computing centers into large scale Grid environments. Finding an effective application performance prediction method for Grid environments constitutes one of particularly important but still open research problems [3, 5, 10]. The aim of the paper is to present an short overview of techniques utilized to predict application execution time. The whole spectrum of prediction methods is analyzed but because of the lack of space the examples of only selected tools dedicated for grid environments are presented.

The paper is organized as follows. Section 2 briefly describes different techniques and metrics used during performance evaluation. The next section is dedicated to the description of different prediction techniques and present the proposed taxonomy. The fourth section shows selected prediction tools. Finally, section 5 outlines the work.

2 Performance evaluation metrics

The performance analysis can be carried out analytically or through experiments. In general, an effective parallel program development cycle, may iterate many times before achieving the desired performance. In parallel programming the goal of the design process is not to optimise a single metrics like for example speed. A good design has to take into consideration a problem specific function of execution time, memory requirements, implementation cost, and others. During performance evaluation of parallel programs different metrics are used [2,6]. The first one is the parallel run time. It is the time from the moment when computation starts to the moment when the last processor finishes its execution. One can find that above definition is ambiguous: what is measured? wall clock time, CPU time or something else. From the other hand we can determine that the parallel run time is composed as an average of three different components: computation time, communication time and idle time, so it can be expressed by the following expression:

$$T_p = \frac{1}{p}\left(\sum_{i=0}^{p-1} T_{comp}^{i} + \sum_{i=0}^{p-1} T_{comm}^{i} + \sum_{i=0}^{p-1} T_{idle}^{i}\right)$$

where p is a number of used processors, the computation time (T_{comp}) is a time spent performing computation by all processors, communication time (T_{comm}) is the time spent for sending and receiving messages by all processors, the idle time (T_{idle}) is when processor stay idle. The last two times are execution overheads specific for parallel programs execution. Moreover different execution overheads related mainly to operating system can appear during parallel as well as serial programs execution. The most of other measured are related to the parallel execution time. The parallel run time of a parallel algorithm depends not only on the size of the problem but also on the complexity of interconnection network and the number of used processor. It means that the parallel runtime is not a good performance metrics for evaluation parallel programs. More convenient metrics is speedup, which captures the relative benefit of solving given problem using parallel system. There exist different speedup definitions. Generally the speedup (S) is defined as the ratio of the time needed to solve the problem on a single processor to the time required to solve the same problem on parallel system with "p" processors. Depending on the way in which sequential time is measured we can distinguish absolute, real and relative speedups. Theoretically, speedup can not exceed the number of processors used during program execution, however, different speedup anomalies can be observed. There are two main laws, which define speedup limitation.

Let's consider the execution time of the parallel program, one can distinguish two program parts: sequential part (P_{seq}) which needs to be executed sequentially using one processor and parallel one ($1\text{-}P_{seq}$) which can be executed independently using number of processors. Let us assume that if we execute this program on a single processor the serial execution time will be t_1. Then if p indicates the number of used processors during parallel execution the parallel run time can be expressed by

$$T_{par} = t_1 * P_{seq} + (1 - P_{seq}) * t_1 / p$$

Then when we divide the serial run time by the parallel run time, speedup will be expressed by the following expression

$$S = \frac{t_1}{t_1 * P_{seq} + (1 - P_{seq}) * t_1 / p} \leq \frac{1}{P_{seq}}$$

From this formulae we can conclude that there exist speedup limitation, for example, if 20% of the program is sequential, then the speedup is lower than 5. Presented result is known as Amdahl's law. The next common used law is Gustafson's law also known as a Gustafson speedup. Let us assume that the execution of the maximum size problem using parallel algorithm with p processors takes $P_{seq} + P_{par} = 1$ time, where P_{seq} indicates the

sequential part of the program and P_{par} the parallel part, respectively. Its sequential time (using one processor) will be $P_{seq} + p*P_{par}$. Then we obtain the following expression that specifies speedup:

$$S_G = \frac{P_{seq} + p * P_{par}}{P_{seq} + P_{par}} = P_{seq} + p * P_{par} \geq p * P_{par}$$

It means that the speedup increases linearly with the number of processors and there is no speedup limitation except the number of processors.

Both above-mentioned performance metrics do not take into account the utilization of processors in the parallel system. While executing a parallel algorithm processors for communication purposes spend some time and some processors can be idle. Then the efficiency of a parallel program is defined as a ratio of speedup to the number of processors. In the ideal parallel system the efficiency is equal to one, in practice efficiency of the parallel systems is between zero and one. The next measure, which is often used in the performance evaluation of parallel programs, is the cost of solving a problem by the parallel system. The cost is usually defined as a product of the parallel run time and the number of processors. The next useful measure is the scalability of parallel system. It is a measure of its capacity to increase speedup in proportion to the number of processors. One can say that a system is scalable when for increasing number of processors and a size of the problem the efficiency is the same. All of the above metrics can be predicted in theoretical or experimental ways that are discussed in the next sections.

3 Performance prediction techniques

The ultimate goal of the application performance analysis and prediction is always answering the question of how the particular application will perform in the particular environment. The parallel application runtime prediction is thus the function of the used hardware, particular software environment, and the application characteristics.

The classification of assumptions that are taken about the hardware, can be seen as largely symmetrical to the ones imposed on the applications. In both the domains a performance prediction technique can have specific requirements over the scope of the solution (hardware or application-related) it can work for. The performance prediction utilizes particular technique to acquire the needed data and employs particular methods to extrapolate the existing information into the final prediction. The software environment domain that the application and the hardware work in is represented by a particular operating system or a parallel processing environment and can be analyzed as the part of the hardware domain.

The assumptions about the hardware and applications, techniques used to acquire the data and the methods of predicting the behavior are the three most important factors determining the accuracy and the scope of utilization of the particular prediction technique. We believe that the characterization of these factors leads to the accurate classification of the prediction method itself. In the following paragraphs the short discussion of the most important possible classes of the three determining factors, together with the examples of actual prediction environments is presented.

3.1 The scope of utilization

A particular prediction environment can have a general scope of utilization, be limited to a given hardware characteristics or applications types or be dedicated to work only for a particular hardware environment or the application. The examples of limitations of hardware characteristics include architecture limitations (e.g. SMP, MPP, Cluster, NOW, Grid, etc) or, when the software environment is analyzed, programming paradigm

limitations (e.g. message passing, shared-memory). In application domain the prediction environment can be dedicated to the general utilization, work only for particular application classes (e.g. only sequential or parallel ones), be dedicated to work for only one application. Naturally more general solutions usually offer less precise predictions. The requirements about the software environment applications are run in (e.g. message-passing or shared-memory programming paradigm) are also the part of the hardware domain scope.

3.2 Data acquisition method

The most precise method in which the data can be collected is the exact theoretical analysis, where the detailed description of the hardware behavior or the application algorithm is performed. For all but the smallest hardware systems the system is too complex to allow the exact hardware behavior analysis to be performed. If this method is not available or feasible, the approximate analysis can be performed. The approximate analysis is based on detailed analysis of only the most important factors (e.g. MIPS, MFLOPS, network bandwidth or latency for hardware and computational complexity and communication patterns for application domain) and approximating the remaining ones. Both of these methods need the presence of the analytical description of the hardware or application they are applied to. The data can also be collected by the measurement of the metrics applicable to the hardware or application behavior. The measurement can be exact, utilizing the dedicated hardware counters and the application code instrumentation or approximate, utilizing the analysis of only the most important characteristics (e.g. wall clock time of the execution or amount of the transferred data for application domain, and the most important hardware characteristics mentioned earlier for approximate analysis). The measuring can be performed in every hardware counter cycle or application step or be conducted with the smaller frequency and the intermediated values interpolated. With the growth of the needed information precision more data has to be collected. This leads not only to the problems with the impact of the measurement on the behavior of the application and the system where it runs but also with the insufficient performance of the needed storage space. The system simulators used for performance evaluation, working in the flexible time domain can overcome the problem but at the cost of the additional time needed to perform the exact simulation or at the cost of the simulation inaccuracy. In hardware domain the simulators are used mainly for “big-iron machines”, the applications execution is simulated either by the execution of the application with only the partial data or limited number of iterations or by running only partially coded version of the program.

3.3 Prediction method

The information about application and system behavior is used to predict the application runtime in the environment conditions where the application was not previously run. The changes to the hardware environment can include only the difference in the number of processors, changes in the number and speed of the processors or general architectural changes. While for the first two types of modifications universal analytical methods exist, the third one is available only in the full system simulation environments. In the application domain the prediction is needed when the application is run with new input data, execution parameters or when it is run on different hardware. The hardware and applications prediction can be based either on the precise analytical calculations or approximations. The precise calculations can be performed only if the full analysis of the application or full hardware specification is present. If either of the conditions is not met, the approximation techniques are needed. The most popular approximation techniques are based on the utilization of the historical data available. If the historical data is present, the prediction is based on the extrapolation of the information to the situation in the new environment. In the more general case the prediction is based on the utilization of either statistical or heuristic techniques (e.g. genetic algorithms or simulated annealing) to find the similarities between analyzed situation and the data previously collected.

3.4 The resulting taxonomy

To summarize the discussion above, the short taxonomy of the most important classes of prediction environments can be proposed. The taxonomy is constructed in a tree-like manner in which a particular prediction technique is described by the selection of a leaf level description for every of the existing classes (i.e. the hardware domain scope, the hardware domain data acquisition method, the hardware domain prediction method, and so forth).

5 Performance prediction tools

Most of existing available performance prediction tools are targeted to application developers, and are helpful during application design, implementation or tuning phase. Still, prediction techniques can be applied in a wide area of issues, especially related to grid computing: the prediction of a resource performance, an application execution or queue wait time. Such predictions can be utilized by load balancing mechanisms, job schedulers, or resource evaluation and selection mechanisms. Below is a brief description of exemplary prediction tools used by grid environments and applications. The attempt is made to classify selected solutions according to the proposed taxonomy. Because of the lack of space only three commonly used different performance prediction tools limited to the grid environment are briefly described below.

5.1 Dimemas Performance Prediction Tool

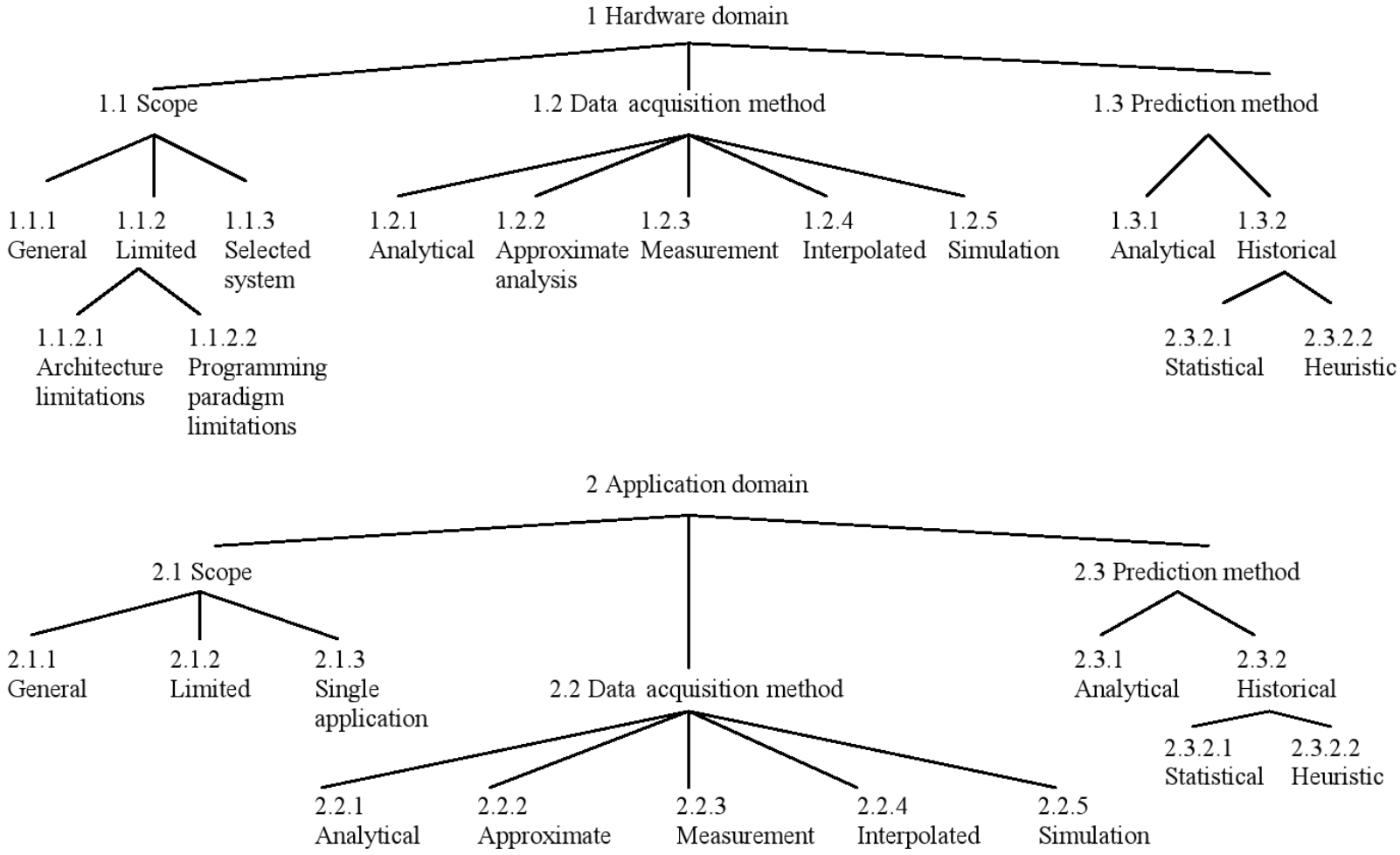
Dimemas is a tool for performance prediction. It uses a model approach and is able to predict execution time for message passing applications run in the Grid. This tool is being used by application developers to ease evaluation of a parallel application in order to gain better performance on different Grid configurations (architectures). The simulation gives the knowledge on how the application will run on different machines with various configurations, together with reasonable estimations of what the performance will be.

The application code should be instrumented using VAMPIRTrace (the previous incarnation of the Intel Trace Analyzer and Collector) libraries. The trace file can be generated on any machine. Dimemas then creates a new time characteristics of parallel application, using the trace file and a configuration file. The configuration file contains *a priori* knowledge about target resource architecture. The output of this process may be a predicted execution time value or a trace file for a visualization and further analysis.

The target architecture of the resources is a Grid modeled as a set of machines connected with WAN. Each machine is considered as a network of SMP nodes, and each node has a set of processors and amount of local shared memory. The overall architecture is described also by a set of performance parameters, e.g. network link latency, bandwidth between processors and memory, between nodes, and machines. The precise architecture model can be found in [1].

Dimemas describes communication patterns of the application using two linear performance models, one for point to point communication and second for global, collective communication. For both types of communication two levels are considered: local (processes on the same machine) and remote (processes on different machines). For the local level there is a further distinction on the node level and SMP level. The models define how the time of every communication step is calculated [1] concerning communication on different levels.

Although the tool seems to be a good solution for application tuning on static environments, there is a possible lack of functionality for dynamic environments, where the parameters and availability of the communication resources may vary in time. However the tool can be used by the grid middleware, for example by scheduling module as on-line prediction tool.


1 Hardware domain
1.1 Scope
1.2 Data acquisition method
1.3 Prediction method
1.1.1 General
1.1.2 Limited
1.1.3 Selected system
1.1.2.1 Architecture limitations
1.1.2.2 Programming paradigm limitations
1.2.1 Analytical
1.2.2 Approximate analysis
1.2.3 Measurement
1.2.4 Interpolated
1.2.5 Simulation
1.3.1 Analytical
1.3.2 Historical
2.3.2.1 Statistical
2.3.2.2 Heuristic
2 Application domain
2.1 Scope
2.2 Data acquisition method
2.3 Prediction method
2.1.1 General
2.1.2 Limited
2.1.3 Single application
2.2.1 Analytical
2.2.2 Approximate
2.2.3 Measurement
2.2.4 Interpolated
2.2.5 Simulation
2.3.1 Analytical
2.3.2 Historical
2.3.2.1 Statistical
2.3.2.2 Heuristic

Dimemas works for a class of hardware limited to Grids, Clusters, NOW and networks of SMPs. In application domain scope it can be used only for message passing applications. In hardware domain it utilizes approximate analysis and in application domain initially employs measurement (application code instrumentation), and the approximate analysis. In both the domains Dimemas uses an analytical method of prediction.

5.2 Network Weather Service

The Network Weather Service (NWS) [9] is a distributed system that periodically monitors and dynamically forecasts the performance that various network and computational resources can deliver to an application over a given time period. Predictions delivered by NWS may be used by middleware tools to improve scheduling decisions or to guarantee required quality of service in a distributed environment. NWS API may also be used directly by grid applications (C, C++, and Java) to get on-line predictions from the system in order to implement their own load balancing mechanisms.

NWS consists of two subsystems – the sensory subsystem and the forecasting subsystem. The sensors are distributed in the monitored environment. Each sensor periodically takes a performance measurement from a monitored resource (cpu usage, memory usage, network link latency, throughput) and stores it with a timestamp in the database. The measurements for a given resource form time series, which describe the behavior of the resource in the past. Based on these characteristics the forecasting subsystem generates a prediction of performance for each resource in a given time frame. There are several statistical prediction methods available in a forecasting subsystem: mean based, median based, and autoregressive. A new forecast is generated by all of the methods after each new measurement, using all the history data collected for the measured resource.

NWS works only in the hardware domain so the application domain characteristics are not applicable here. It has a general scope but the most uses are for Clusters and Grids. Data acquisition method is based on periodic measurements. NWS uses statistical methods of forecasting (mean, median, autoregressive).

5.3 Grid performance PREdiction System

The grid performance PREdiction System (GPRES) provides information about possible application queue wait time and execution times on given grid resources. As described in [7] GPRES is constructed as an advisory system for resource brokers managing distributed environment, including computational Grids. The system is designed to work with any resource management system, but was initially adapted to cooperate with the Grid Resource Management System (GRMS) decision module. The system is implemented as an expert system. With this approach the knowledge acquisition can be separated from the prediction process.

The knowledge acquisition mechanisms use historical information about the environment and application runs. The technique assumes that the information about historical jobs can be used to predict new job times characteristics. GPRES system uses the “similarity templates” approach to determine similar jobs [4, 8]. The template is a set of job attributes, which are used to evaluate jobs’ similarity. The assumption is made, that similar jobs have similar execution time thus the time of a new job is estimated based on previously run similar jobs. The historical jobs are categorized according to templates and the mean value of its execution times is taken as estimation. To determine the set of templates the system is working with, the Coefficient of Variation (CV) value calculated for each template may be used or clustering algorithms.

Historical data is gathered by Data Providers, small components distributed in the Grid. There are several providers available, for LSF, PBS and LoadLeveler. One provider is adapted to collect information from broker’s logs. The Data preprocessing module prepares data for knowledge acquisition. Jobs parameters are unified and joined, specific application and system data are processed. Such prepared data is used by the Knowledge Acquisition Module to generate rules. Rules are inducted into Knowledge Database. When the estimation request comes to GPRES the Request processing module prepares all the incoming job and resource data for the reasoning module. Reasoning module selects rules from Knowledge Database and generates requested estimation. There are several strategies of selecting the best rule available in GPRES, e.g.: choose the most

specific rule, choose the rule generated from the highest amount of history jobs. Some heuristics are used to combine these strategies.

The tool has a well defined interfaces, and can be adapted to many grid environments. Although, the system should be extended to use information published by standard grid information systems rather than be dedicated own data providers.

GPRES works for Grid systems, uses historical method of hardware data collection and historical (both statistical and heuristic) methods of hardware conditions prediction. In the application domain it works for general class of applications, uses historical information gathered from local and grid management systems and uses statistical and heuristic based methods of historical prediction.

6 Conclusions

In the paper a brief discussion related to available performance prediction tools is presented. We can conclude that the problem of finding an effective application performance prediction method is still open. The current trends in parallel and distributed processing make it particularly important for grid computing. Such predictions can be utilized by load balancing mechanisms, job schedulers, or resource evaluation and selection mechanisms. Unfortunately most of the existing available performance prediction tools are targeted to some specific environments (hardware, software) or even to specific application. Additionally in practice, there are no tools which can be used on-line during application execution. We hope that the discussion presented in the paper is a small step in the research concerned with finding a general method of performance prediction that can be used for off-line as well as on-line tools.

Acknowledgements: This research was partially supported by the European Community Framework Programme 6 project DeDiSys, contract No 004152.

References

1. Badia R.M., Labarta J., Gimenez J, Escale F., *DIMEMAS: Predicting MPI applications behavior in Grid environments,* Workshop on Grid Applications and Programming Tools (GGF8), 2003.
2. Foster I., *Designing and Building Parallel Programs* , Addison-Wesley Pub., 1995 (also available at `http://www.mcs.anl.gov/dbpp/text/book.html`).
3. Gerndt M., Wismueller R. & others, *Performance Tools for the Grid: State of the Art and Future,* Research Report Series, Lehrstuhl fuer Rechnertechnik und Rechnerorganisation (LRR-TUM), vol. **30** , Shaker Verlag, 2004.
4. Gibbons, R.: *A Historical Application Profiler for Use by Parallel Schedulers,* In: Job Scheduling Strategies for Parallel Processing. Springer Verlag, 1997.
5. Krauter K., Buyya R., Maheswaran M., *A taxonomy and survey of grid resource management systems for distributed computing,* Software Practice and Experience, 2002.
6. Kwiatkowski J., *Performance Evaluation of Parallel Programs, Proceedings of the International Conference Parallel Processing and Applied Mathematics PPAM´99*, Kazimierz Dolny, Poland 1999, pp. 75-85.
7. Kurowski K., Oleksiak A., Nabrzyski J. & others, *Multi-criteria Grid Resource Management Using Performance Prediction Techniques,* CoreGrid Integration Workshop, Pisa, Italy, November 2005, Springer CoreGRID Proceedings, vol. **4**.
8. Smith W., Taylor V., Foster I., *Using run-time predictions to estimate queue wait times and improve scheduler performance,* In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), LNCS, Springer Verlag, 1999.
9. Wolski R., Spring N., Hayes J., *The network weather service: A distributed resource performance forecasting service for metacomputing,*. Future Generation Computer Systems, **15** (5-6), 1999.
10. Yu J., Buyya R., *A taxonomy of scientific workflow systems for grid computing,* ACM SIGMOD Record, **34**(3), 2005.