

# Optimizing Performance of Automatic Training Phase for Application Performance Prediction in the Grid

Farrukh Nadeem, Radu Prodan, and Thomas Fahringer

Institute of Computer Science, University of Innsbruck  
Technikerstraße 21a, A-6020 Innsbruck, Austria  
{farrukh,radu,tf}@dps.uibk.ac.at

**Abstract.** Automatic execution time prediction of the Grid applications plays a critical role in making the pervasive Grid more reliable and predictable. However, automatic execution time prediction has not been addressed due to the diversity of the Grid applications, usability of an application in multiple contexts, dynamic nature of the Grid, and concerns about result accuracy and time expensive experimental training. We introduce an optimized, low-cost, and efficient yet automatic training phase for automatic execution time prediction of Grid applications. Our approach is supported by intra- and inter-platform performance sharing and translation mechanisms. We are able to reduce the total number of experiments from an polynomial complexity to a linear complexity.

## 1 Introduction

Automatic execution time prediction of the Grid applications plays a critical role in making the pervasive Grid more reliable and predictable. Application execution time prediction based on historical data is a generic technique used for application performance prediction of scientific and business applications in the Grid infrastructures [14]. Historical data obtained through carefully designed experiments can minimize the need and cost of performance modeling. Controlling number of experiments to get the historical/training data, in order to control the complexity of training phase is challenging. Methods from the field of experimental design have been applied to similar control problems in many scientific and engineering fields for several decades. However, experimental design to support automatic training phase and performance prediction in the Grid has been largely ignored due to diversity of the Grid applications, usability of applications in different contexts and heterogeneous environments, and concerns about the result accuracy and time expensive experimental training. There may be some applications whose scientific phenomenon of performance behavior could be well understood, and useful results including performance models can be developed, but, this is not possible for all applications because of expensive performance modeling, which makes it almost impractical to serve several predictions using these models at run time. Moreover, the size, diversity, and

extensibility of heterogeneous resources make it even harder for the Grid. These challenging issues lead to the requirement of a generic and robust experimental design for automatic training phase that maximizes the amount of information gained in minimum number of experiments by optimizing the combinations of independent variables.

To overcome this situation, we present a generic experimental design EXD (Extended Experimental Design) for compute intensive applications. EXD is formulated by modifying the traditional experimental design steps to eliminate time taking modeling and optimization steps that make it inefficient and application specific. We also introduce inter- and intra-platform performance sharing and translation mechanisms to later support EXD. Through EXD, we are able to reduce the polynomial complexity of training phase to a linear complexity, and to show up to a 99% reduction in total number experiments for three scientific applications, while maintaining an accuracy of more than 90%. We have implemented a prototype of the system based on the proposed approach in ASKALON project [12], as a set of Grid services using Globus Toolkit 4.

The rest of the paper is organized as follows: Section 2 describes an introduction to Automatic Training Phase and System Architecture of automatic training phase is presented in Section 3. Section 4 describes the performance sharing and translation mechanisms, which support our design of experiments. Section 5 narrates design and composition of EXD. Section 6 explains the Automatic Training phase based on EXD. Application performance prediction based on the training set is described in Section 7, and experimental results with analysis are summarized in Section 8. Finally we describe related work in Section 9.

## 2 Training Phase on the Grid

The training phase for an application consists of the application executions for different setups of problem-size and machine-size on different Grid-sites, to obtain execution times (training set or historical data) for those setups. Automatic performance prediction based on historical data needs *enough* amounts of data present in database. The historical data needs to be generated for every new application ported to a Grid environment, and/or for every new machine (different from existing machines) added to the Grid. To support the automatic application execution time prediction process experiments should be made against some experimental design and the generated training data be archived automatically. However, to automatize this whole process is a complex problem and we address it in Section 2.1. Here, we describe some of the terms that we use in this paper.

- *Machine-size*: The total number of processors on a Grid site.
- *Grid-size*: The total number of Grid-sites in the Grid.
- *Problem-size*: Set of input parameter(s) effecting performance of application.

### 2.1 Problem Description

Conducting an automatic training for application execution time predictions on the Grid is a complex problem due to complexity of the factors involved in it.

Generally speaking, automatic training phase is the execution of  $E$  experiments, for a set of applications  $\alpha$ , on a selected set of Grid sites  $\beta$  each with a scarce capacity (e.g. number of processors  $p_i$ ), for a set of problem-sizes  $\lambda$ . Execution of each experimental  $e \in E$  for a Grid application  $\alpha \in \alpha_i$ , for a problem-size  $r \in \lambda$ , on a Grid site  $g \in \beta$ , at a certain site capacity  $p \in \gamma$  yields the execution time interval (duration)  $T(e) = endt(e) - startt(e)$ . More formally, the automatic training phase comprises of:

- A set of Grid applications:  $\alpha = \{\alpha_1, \dots, \alpha_n\} | n \in \mathbb{N}$ .
- A set of selected non-identical Grid sites:

$$\beta = \cup_{i=1}^m g_i \mid \beta \subseteq G_T, |\beta| = m$$

where  $G_T$  represents set of all the Grid sites. Thus the total Grid capacity in terms of processors is  $\gamma = \sum_{i=1}^m p_i$ , where  $p_i$  is number of processor on the Grid-site  $i$ .

- A set of different machine-sizes  $\omega$ , encompassing different machine sizes for  $z$  different (heterogeneous) types of CPUs on each of the Grid-sites  $g_i$ , where  $\omega_i = \cup_{k=1}^z \{1, \dots, S_k\}$ . Here  $S_k$  represents maximum number of CPUs of type  $k$ . We categorize CPUs to be different if their architecture and/or speed is/are different.
- A set of problem-sizes  $\lambda = \{r_1, r_2, \dots, r_x\} \mid r_i = \cup_{j=1}^y param_j, |\lambda| = x$ , here  $param_j$  represents value of an input parameter  $j$  of the Grid application  $\alpha$  having  $y$  parameters in total, which depends upon the Grid application.

In this way, for  $n$  applications,  $m$  Grid-sites,  $z_i$  different types of CPUs on a Grid-site  $i$  where maximum number of CPUs of category  $k$  is  $S_k$ , the total number of experiments  $N$  is given by:

$$N = \sum_{h=1}^n \sum_{i=1}^m \sum_{k=1}^{z_i} \sum_{s=1}^{S_k} p_{k,s}^i \times x_h$$

Where  $p_{k,s}^i$  denotes  $s$  processors of type  $k$  on Grid-site  $i$ , and  $x_h$  denotes the number of different problem sizes of application  $h$ .

The size of each of  $\alpha, \beta, \gamma, \lambda$  and  $\omega$  has a significant effect on the overall complexity of the automatic training phase. The goal is to come up with a set of execution times from  $E$  experiments  $\psi : \psi = \{t_1, \dots, t_E\} \mid |\psi| < N$ , such that utility  $U$  of execution times  $\sum_{e=1}^E U(t_i)$  is maximized,  $N$  is minimized, and the accuracy  $\xi$  is maximized.

### 3 System Architecture

The Architecture of the system is simple and shown in Figure 1. Application specific information (the different parameters affecting execution time of the application, their ranges and effective step sizes) is provided to Experiment Design Manager, which plans experiments through Extended Experimental Design

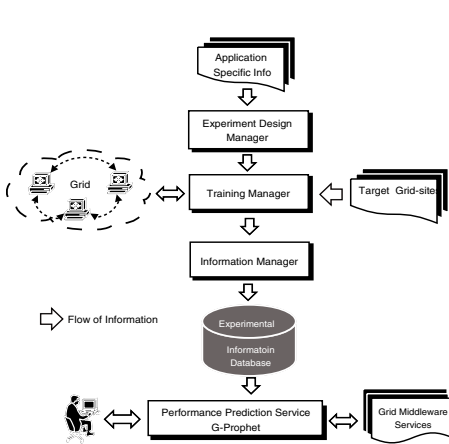


Fig. 1. System Architecture

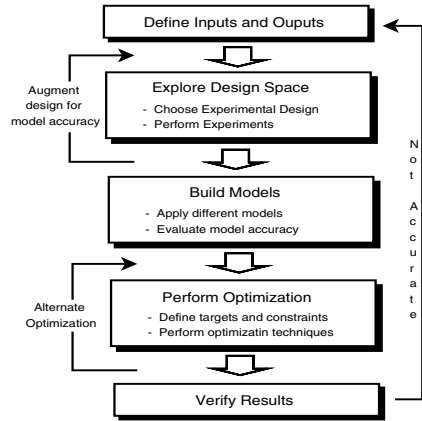


Fig. 2. Sequence of steps in a typical experimental design

(EXD) (see Section 5). The Training Manager executes these experiments on the set of the selected Grid sites. It dynamically decides the detailed execution of minimum number of experiments on different Grids-sites. The information Manager orchestrates the experimental information and, according to configuration policy, either archives this information directly in experimental performance repository or uses the performance sharing and translation mechanisms (see Section 4) to calculate the remaining values (not in the experimental information) before archiving the information. The prediction engine of application execution time prediction service (G-Prophet) gets the archived information from performance repository to serve application execution time predictions to different Grid middleware services.

## 4 Performance Sharing and Translation (PST)

We introduce PST to share execution times within one Grid-site (for scalability) and among different Gridsites. PST mechanism is based on our experimental observation of inter- and intra-platform performance relativity (rate of change of performance) of compute intensive applications, across different problem-sizes. We explain them as under.

### 4.1 Inter-platform PST

Inter-platform PST specifies that the normalized execution time  $\rho$  of an application for the different problem-sizes in  $\alpha$ , is preserved on different Grid sites. More specifically, the normalized execution time  $\rho$  for a problem-size  $r_i$  relative to another problem-size  $r_j$  on a Grid-site  $g$  is similar to that on another Grid-site

$h$ . If  $\rho_g(\alpha, r_i)$  represents the execution time of application  $\alpha$  for problem-size  $r_i$  on Grid-site  $g$  then:

$$\frac{\rho_g(\alpha, r_i)}{\rho_g(\alpha, r_j)} \approx \frac{\rho_h(\alpha, r_i)}{\rho_h(\alpha, r_j)}, i \neq j \quad (1)$$

This phenomenon is based on the fact that rate of change in execution time of an application across different problem-sizes is preserved on different Gridsites, i.e. the rate of change in execution time of an application  $\Delta\rho$  for the problem-size  $r_i$  (the target problem-size) with respect to another problem-size  $r_j$  (the reference problem-size) on Grid-site  $g$  is equal to the rate of change in execution time for the problemsize  $r_i$  with respect to the problem-size  $r_j$  on Grid-site  $h$ .

$$\frac{\Delta\rho_g(\alpha, r_i)}{\Delta\rho_g(\alpha, r_j)} \approx \frac{\Delta\rho_h(\alpha, r_i)}{\Delta\rho_h(\alpha, r_j)}, i \neq j$$

## 4.2 Intra-platform PST

Intra-platform PST specifies that the normalized execution time  $\rho$  of an application  $\alpha$ , on a Grid-site  $g$  for a machine-size  $l \in \omega_g$  (the target machine-size) relative to another machine-size  $m \in \omega_g$  (the reference machine-size), for a problem-size  $r_i$  is similar that for another problem-size  $r_j$ . If  $\rho_g(\alpha, r_i, l)$  represents the execution time of an application for problem-size  $r_i$  and machine-size  $l$  then:

$$\frac{\rho_g(\alpha, r_i, l)}{\rho_g(\alpha, r_j, m)} \approx \frac{\rho_g(\alpha, r_i, l)}{\rho_g(\alpha, r_j, m)}, i \neq j, l \neq m \quad (2)$$

This phenomenon is based on the fact that rate of change in execution time of an application across different problem-sizes is preserved for different machine-sizes, i.e. the rate of change in execution time of an application for the problem-size  $r_i$  and machinesize  $l$  on Grid-site  $g$  with respect to that for machinesize  $m$  will be equal to the rate of change in execution time for the problem-size  $r_j$  and machine-size  $l$  with respect to that for a machine-size  $m$  on the same Grid-site:

$$\frac{\Delta\rho_g(\alpha, r_i, l)}{\Delta\rho_g(\alpha, r_j, m)} \approx \frac{\Delta\rho_g(\alpha, r_i, l)}{\Delta\rho_g(\alpha, r_j, m)}, i \neq j, l \neq m$$

Similarly, rate of change in executions time of the application across different machine sizes is also preserved for different problem-sizes. i.e.

$$\frac{\Delta\rho_g(\alpha, r_j, m)}{\Delta\rho_g(\alpha, r_i, m)} \approx \frac{\Delta\rho_g(\alpha, r_j, l)}{\Delta\rho_g(\alpha, r_i, l)}, i \neq j, l \neq m$$

We use this phenomenon to share execution times with in one Grid-site for scalability. The accuracy of inter- and intra-platform similarity of normalized behaviors, for embarrassingly parallel applications, does not depend upon the selection of reference point. However, for the parallel applications exploiting inter-process communications during their executions, this accuracy increases as the reference point gets closer to the target point the closer the reference point,

the greater the similarity (of interprocess communication) it encompasses. Thus in case of inter-platform PST the reference problem-size closer to the target problem-size, and in case of intra-platform PST, the reference problem-size closer to the target problem-size as well as the reference machine-size closer to target machine-size. i.e. For inter-platform PST:

$$\lim_{r \rightarrow p} \left[ \frac{\rho_g(\alpha, r_i)}{\rho_g(\alpha, r_j)} - \frac{\rho_h(\alpha, r_i)}{\rho_h(\alpha, r_j)} \right] = 0$$

Similarly, for intra-platform PST:

$$\lim_{l \rightarrow m} \left[ \frac{\rho_g(\alpha, r_i, l)}{\rho_g(\alpha, r_j, m)} - \frac{\rho_h(\alpha, r_i, l)}{\rho_h(\alpha, r_j, m)} \right] = 0$$

For normalization from the minimum training set only, we select the maximum problem size (in normal practice of user of the application) and maximum machine size (for which application scales good) as reference point, to incorporate the maximum effects of inter-process communications in the normalization. The distance between the target point and the reference point for inter- and intra-platform PST on one Grid site is calculated respectively as:

$$d = \begin{cases} \sqrt{(\rho(r_i) - \rho(r_j))^2 + (r_i - r_j)^2} \\ \sqrt{(\rho(l) - \rho(m))^2 + (l - m)^2} \end{cases}$$

Note that, in the presented work,  $\rho_g(\alpha, p) = \rho_g(\alpha, p, 1)$ .

## 5 Experimental Design

Specifically in our work, the general purpose of the experimental design phase is to set a strategy for experiments to get the execution time of an application to support its performance prediction later on, in minimum number of experiments. A typical experimental design has specific sequence of steps, which is shown in Figure 2. For experimental design, one of our objectives is to eliminate/minimize the modeling and optimization phases in this model to make the training phase and performance prediction system robust. Among other key objectives are, to:

- a) reduce/minimize training phase time;
- b) minimize/eliminate the heavy modeling requirements after the training phase;
- c) develop and maintain the efficient scalability of ED with respect to Grid-size;
- d) make it generalizable to a wide range of applications on heterogeneous Grid-sites.

To address these objectives, we design our experimental design EXD in the light of guide lines given by Montgomery et al. in [1]. These are as follows:

**a) Recognition of statement of problem:** We describe our problem statement as: To obtain maximum execution time information of the application at

different problem-sizes on all heterogeneous Grid-sites with different possible machine-sizes in minimum number of the experiments.

**b) Selection of response variables:** In our work the response variable is the execution time of the application.

**c) Choice of factors, levels and ranges:** The factors affecting the response variable are the problem-size of the application, the Grid-size and the machine-size. The range of problem-size incorporates ranges of each of input variables and the levels consist of their values at effective step sizes specified by the user of the application. The range of the Grid-size is  $\{1, 2, \dots, m\}$ , and the levels include numeration of non-identical Grid-sites  $g_i$ . The range of machine-size spans over all different subsets of number of (heterogeneous) processors on a Grid-site and its levels include of all these individual subsets. Consider a simple case, for one Grid application with  $x$  different problem sizes and  $m$  Grid sites with  $p_i$  machine sizes. If we include all the respective levels of the above described factors, then the total number of possible experiments  $N$  is given by:

$$N = \sum_i^m (p^i \times x)$$

For single processor Grid-sites the above formula reduces to  $N = x \times m$ . The objective function  $f$  of the experimental design is:

$$f : \mathbb{R} \times \mathbb{N}^+ \times \mathbb{N}^+ \rightarrow \mathbb{N}^+ \\ f(\beta, \gamma, \lambda) = \text{Min}_N : \xi \geq 90\%$$

where  $\xi$  represents accuracy.

**d) Choice/Formulation of Experimental Design:** In our Extended Experimental Design (EXD), we minimize the combinations of Grid-size with problem-size and then, combinations of Gridsize with machine-size. By minimizing the combinations of Grid-size with problem-size, we actually minimize number of experiments against different problem-sizes across the heterogeneous Grid-sites. Similarly, by minimizing the Grid-size combinations with machine-size factor, we actually minimize number of experiments against different problem-sizes across different number of processors. To meet these objective, we employ PST [2] mechanisms (see Section 4.2).

In our design, we choose one Grid-site (the fastest one, based on the initial runs) as a base Grid-site and make a full factorial of experiments on it. Later, we use its execution times as reference values to calculate predictions for other platforms (the target Grid-sites) using inter-platform PST and thus minimize problem-size combinations with Grid-size. We also make one experiment on each of other different Grid-sites to make PST work. Similarly, to minimize machine-size combinations with Grid-size, we need a full factorial of experiments with one machine-size, the base machine-size (which we already have while minimizing problem-size combinations with Grid-size), and later use these values as reference values. We make one experiment each for all other machine-sizes, to translate the reference execution times to that for other machine-sizes using

intraplatform PST. The approach of making full factorial design of experiments is also necessary because we neither make any assumptions about the performance models of different applications, nor we make their analytical models at run time, because making analytical performance models for individual applications is more complex and less efficient than making full factorial design of experiments for once. In initial phase, we restrict the full factorial design of experiments to the commonly used range of problem-size.

By means of inter-platform PST, the total number of experiments  $N$  reduces from a polynomial complexity of  $p \times x \times m$  to  $p \times x + (m - 1)$  for parallel machines, and from a polynomial complexity of  $x \times m$  to a linear complexity of  $x - 1 + m$  for single processor machines. Introducing intra-platform PST, we are able to reduce total number of experiments for parallel machines (Grid-sites) further to a linear complexity of  $p + (x - 1) + (m - 1)$  or  $p + m + x - 2$ .

**e) Performing of experiments:** We address performing of experiments under automatic training phase as described in Section 6.

We design above step d to eliminate the need of next two steps presented by Montgomery et. al. the *statistical analysis & modeling* and *conclusions*, to minimize the serving costs on the fly.

## 6 Automatic Training Phase Based on EXD

If we consider the whole training phase as a process then, in executing the designed experiments, the process variables are application problem-size, machine-size and Grid-size. Our automatic training phase is a three layered approach; layer 1 for the initial test runs, layer 2 for the execution of experiments designed in experimental design phase, and layer 3 for the PST mechanism. In the initial test runs of the training phase, we need to make some characterizing or screening to establish some conjectures for the next set of experiments. This information is used to decide the base Grid-site. In layer 2, execution of experiments on different Grid-sites is planned according to the experimental design, and the training manager executes these experiments on the selected Grid-sites. The collected information is passed through layer 3, the PST mechanism, to store in a performance repository. First, the training manager makes one initial experiment on all non-identical Grid-sites  $G$  from Grid-site sample space  $G_T$  for maximum values of problem-sizes. Second, it selects the Gridsite with minimum execution time as base Grid-site and makes a full factorial design of experiments on it. We categorize two Grid-sites to be identical if they have same number of processors, processor architecture, processor speed, and memory. We also exclude those identical Grid-sites with  $i$  processors  $g_i$  for which  $g^i \subset g^j$  for  $i \leq j$ , i.e. if a cluster or a sub-cluster is identical to another cluster or sub-cluster then only one of them will be included in training phase. Similarly, if a cluster or a sub-cluster is a subset of another cluster or sub-cluster then only the super set cluster is included in the training phase

The information from these experiments is archived to serve prediction process. For applications having a wide range of problem-size, we have support of

distributed training phase. In distributed training phase we split the full factorial design of experiments from one machine to different Grid-sites which are included in the training phase. Distributed training phase exploits opportunistic load balancing [15] for executing experiments to harness maximum parallelization of training phase.

## 7 Application Performance Prediction System: G-Prophet

The ultimate goal of automatic training phase is to support performance prediction service, to serve automatic execution time predictions. To serve application performance prediction prediction system G-Prophet (Grid- Prophet), implemented under ASKALON, utilizes the inter- and intra-platform PST mechanisms to furnish the predictions using Equation 1 as:

$$\rho_g(\alpha, r_i) = \frac{\rho_h(\alpha, r_i)}{\rho_h(\alpha, r_j)} \times \rho_g(\alpha, r_i)$$

and/or Equation 2 as:

$$\rho_g(\alpha, r_i, l) = \frac{\rho_g(\alpha, r_i, l)}{\rho_g(\alpha, r_j, m)} \times \rho_g(\alpha, r_i, m)$$

G-Prophet uses the the nearest possible reference values for serving the predictions, available from the training phase or actual run times. From the minimum training set, we observe a prediction accuracy of more than 90% and a standard deviation of 2% in the worst case.

## 8 Experiments and Analysis

In this section, we describe the experimental platform and real world applications used for our experiments, the scalability and performance results from our proposed experimental design against the different changes in the factors involved (as defined in Section 5). For our results presented here, each result is taken as an average of five repetitions of the experiment to guard against the anomalous results.

### 8.1 Experimental Platform

We have conducted all of our experiments on the Austrian Grid environment. The Austrian Grid consortium combines Austria's leading researchers in advanced computing technologies with well-recognized partners in Grid-dependant application areas. The description of Austrian test bed sites for our experiments is given in Table 2. Our experiments are conducted on three real world applications Wien2k [3], Invmod [2] and meteoAG [4]. Wien2k application allows performing electronic structure calculations of solids using density functional theory based on the full-potential augmented planewave ((L)APW) and

local orbital (lo) method. Invmold application helps in studying the effects of climatic changes on the water balance through water flow and balance simulations, in order to obtain improved discharge estimates for extreme floods. MeteoAG produces meteorological simulations of precipitation fields of heavy precipitation cases over the western part of Austria with RAMS, at a spatially and temporally fine granularity, in order to resolve most alpine watersheds and thunderstorms.

## 8.2 Performance and Scalability Analysis

The EXD scales efficiently against the changes in different factors involved in ED. During our experiments we obtain quite promising results. The scalability analysis is shown in Figure 3. We analyzed the scalability of EXD w.r.t. problem-size, by varying the problem-size factor for fixed remaining factors; 10 parallel Grid-sites with machine-size 20 and 50 single processor machines. The problem-size was varied from 10 to 200 and the reduction in the total number of experiments was observed from 96% to 99%. In another setup for analyzing scalability w.r.t machine-size, a reduction of 77% to 97% in the total number of experiments was observed when machine-size was varied from 1 to 80, for fixed factors of 10 parallel machines, 50 single processor Grid-sites and problem-size of 5. From another perspective, we observed that total number of experiments increased from 7% to 9% when Grid-size was increased from 15 to 155, for the fixed factors of 5 parallel machines with machine-size of 10 and problem-size 10. We observed an overall reduction of 78% to 99% when all the factors we varied simultaneously: 5 parallel machines with machine-size from 1 to 80, single processor Grid-sites from 10 to 95, and problem-size from 10 to 95. Performance and normalization results for interplatform PST are shown for Wien2k and Invmold in Figure 4, 5 and 6, 7 respectively. Figure 4 shows the measured performance of Wien2k on 5 different Grid-sites, and Figure 5 shows the normalized performance on these Grid-sites. For Wien2k the normalization is made with the execution time against the problem-size of 9.0. Figure 6 shows the measured execution time of Invmold on 5 different Grid-sites, and Figure 7 shows the normalized performance on these Grid-sites. The normalization for Invmold is made with execution time against the problem-size of 20.0. Automatic training phase made only 49 experiments out of total 492 for Wien2k (approx. 10%), and for Invmold conducted only 24 experiments out of total 192 (approx. 14%) on the test bed described above. Identical curves of normalized execution times exhibit the realization of inter-platform PST for these applications. Performance normalization results for interplatform PST are shown in Figure 8, 9 and 10, 11. Figure 8 and 10 show the measured performance of MeteoAG for different values of problem-sizes and machine-sizes on two different Grid-sites hcma and zid-cc. Figures 9 and 11 show normalized performance of MeteoAG on these Grid-sites. The normalization is performed with the execution time against machine-size of 1. The training phase conducted only 162 out of total 6669 experiments (approx. 2.2%). The identical normalized performance curves show the realization of inter-platform PST.

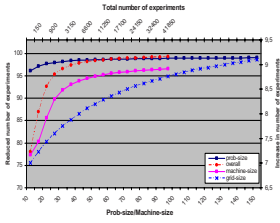


Fig. 3. Scalability analysis of EXD

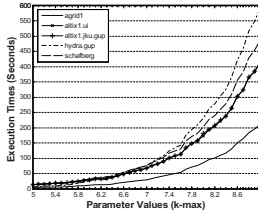


Fig. 4. Wien2k original exe. times

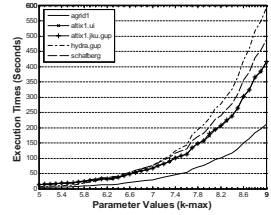


Fig. 5. Wien2k original exe. times

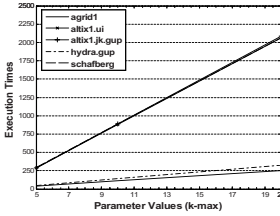


Fig. 6. Invmold original exe. times

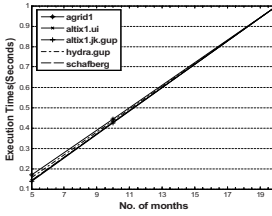


Fig. 7. Invmold normalized exe. times

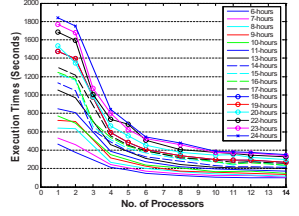


Fig. 8. MeteoAG on *hcma*, original exe. times

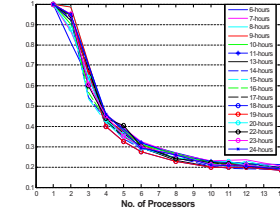


Fig. 9. MeteoAG on *hcma*, normalized exe. times

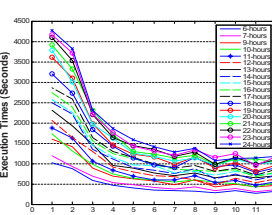


Fig. 10. Invmold normalized exe. times

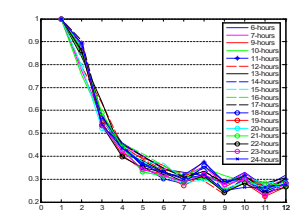


Fig. 11. MeteoAG on *hcma*, original exe. times

## 9 Related Work

There have been some attempts to get reactive training set for performance prediction of Grid application through Analytical benchmarking and templates [5,6], but to our knowledge we are the first to attempt to make a proactive training phase with proper experiment management and control through a step by step experimental design. Many multi-platform performance studies like [11] evaluated their approaches with data collected at multiple Grid-sites. However, data from each Grid-site are processed individually. Our approach, instead, utilizes benchmark performance results from one platform (base Grid-site) to share this information for other platforms (target Grid-sites). Iverson et al. [5] and

Marin et al. [13] use parameterized models to translate execution times across the heterogeneous platforms but their models need human interaction to parameterize all machines performance attributes. In contrast, we make our models on the basis of normalized/relative performance, which does not need any source code and manual intervention. Moreover it is difficult to use their approach for different languages. Leo et al. [7] use partial executions for cross platform performance translation. Their models need source code instrumentation and thus require human intervention. To contrast, our approach does not require source code and code instrumentation. Systems like Prophecy [9], PMaC [8], and convolution methods to map hand-coded kernels to Grid-site profiles [10] base on modeling computational kernels instead of complex applications with diverse tasks. In contrast, we develop observation-based performance translation, which does not require in-depth knowledge of parallel systems or codes. This makes our approach application-, language- and platform-independent. Reducing the number of experiments in adaptive environments dedicated to applications, by tuning parameters at run time have been applied by couple of works like ATLAS [15] (tuning performance of libraries by tuning parameters at run time). Though this technique improves the performance, yet is very specific to the applications. Unlike this work, we do not do run time modeling/tuning at run time, as modeling individual applications is more complex and less efficient than making the proposed experiments.

## References

1. Douglas, C., Montgomery: Design and Analysis of Experiments. ch. 1, vol. 6. John Wiley & Sons, New York (2004)
2. Theiner, D., et al.: Reduction of Calibration Time of Distributed Hydrological Models by Use of Grid Comp. and Nonlinear Optimisation Algos. In: International Conference on Hydroinformatics, France (2006)
3. [3] Blaha, P., Schwarz, K., Madsen, G., Kvasnicka, D., Luitz, J.: WIEN2k: An Augmented Plane Wave plus LocalOrbitals Program for Calculating Crystal Properties, Institute of Physical and Theoretical Chemistry, TU Vienna (2001)
4. Schller, F., Qin, J., Farrukh N.: Performance, Scalability and Quality of the Meteorological Grid Workflow MeteoAG. In: 2nd Austrian Grid Symposium, Innsbruck, Austria. OCG Verlag (2006)
5. Iverson, M.A., et al.: Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment. In: Heterogeneous Computing Workshop (1999)
6. Smith, W., Foster, I., Taylor, V.: Predicting Application Run Times Using Historical Information. In: Proceedings of the IPPS/SPDP (1998) Workshop on Job Scheduling Strategies for Parallel Processing (1998)
7. Yang, L.T., Ma, X., Mueller, F.: Cross- Platform Performance Prediction of Parallel Applications Using Partial Execution. In: Supercomputing, USA (2005)
8. Carrington, L., Snavely, A., Wolter, N.: A performance prediction framework for scientific applications, Future Gener. Computing Systems, vol. 22, Amsterdam, The Netherlands (2006)

9. Taylor, V., et al.: Prophecy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications. *ACM Sigmetrics Performance Evaluation Review* 30(4) (2003)
10. Bailey, D.H., Snaveley, A.: Performance Modeling: Understanding the Present and Predicting the Future. In: *Euro-Par Conference* (August 2005)
11. Kerbyson, D.J., Alme, H.J., et al.: Predictive Performance and Scalability Modeling of a Large-Scale Application. In: *Proceedings of Supercomputing* (2001)
12. Fahringer, T., et al.: ASKALON: A Grid Application Development and Computing Environment. In: *6th International Workshop on Grid, Seattle, USA* (November 2005)
13. Marin, G., Mellor-Crummey, J.: Cross-Architecture Performance Predictions for Scientific Applications Using Parameterized Models. In: *Joint International Conference on Measurement and Modeling of Computer Systems* (2004)
14. Nadeem, F., Yousaf, M.M., Prodan, R., Fahringer, T.: Soft Benchmarks-based Application Performance Prediction using a Minimum Training Set. In: *Second International Conference on e-Science and Grid computing, Amsterdam, Netherlands* (December 2006)
15. Whaley, R.C., Petitet, A.: Petitet: Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience* 35(2):1012013121 (2005)