Predicting Parameter Sweep Jobs: From Simulation to Grid Implementation

Peter Hellinckx, Sam Verboven, Frans Arickx and Jan Broeckhove
Department of Mathematics and Computer Sciences
University of Antwerp
Antwerp, Belgium
Email: peter.hellinckx@ua.ac.be

Abstract—Efficiently using the computational power made available through desktop grids based distributed systems is a complicated and many-sided problem, caused by the intermittent resource availability. In this paper a novel solution is presented for predicting the runtimes of parameter sweep jobs. These jobs are characterized by their lack of inter-dependence and suitability for runtime prediction by modeling. This makes them ideal candidates for deployment on volatile grid configurations using prediction based techniques. The parameter sweep prediction framework used to make the predictions is referred to as GIPSy (Grid Information Prediction System). Previous research involving GIPSy has focused on results obtained during simulation were it is necessary to make some basic assumptions. By combining GIPSy with PGS (Prediction based Grid Scheduling), an actual grid implementation, real results can be obtained. A detailed comparison between the expected results, based on simulation analysis, and the final results is given. Discrepancies are highlighted and possible causes are identified, solutions are proposed and implemented. By comparing the results for different model building configurations an optimal configuration is found that produces reliable result independent of the chosen job type. Results are presented for a quantum physics problem and two simulated workloads represented by sleepjobs.

I. INTRODUCTION

Due to the ever rising demand for more computational power, researchers are always looking for new and better ways to exploit existing systems. Using existing, user-controlled resources (e.g. workstations, personal computers) in large scale grid systems is one of the most cost effective options to increase the amount of available CPU’s. The owners of the donated resources can be both private users as well as large organizations. By using CPU harvesting, otherwise wasted CPU cycles can be used to build desktop grids. These grids have the advantage that it is relatively easy to increase grid capacity requiring minimal investment. By allowing access to previously unused computational power these grids also present many new challenges. In this paper we concentrate on finding methods to use these resources to their full potential. Possible systems that can be used to build such grids include BOINC [1](cf. Seti@home [2]), CoBRA [3] [4] and Condor [5] [6].

Distributed systems provide unique scale and cost benefits. There are however some significant challenges that need to be taken into consideration when building a desktop grid. The problem that is addressed in this paper has to do with the optimal distribution of tasks within a dynamic grid environment. Since resources are separately owned and managed, each may enter and leave the grid at any given time. The resulting churn can lead to significant overhead. Each aborted job will need to be restarted elsewhere, resulting in a loss of CPU cycles. One of the possible options to counter this inherent side effect of desktop grids is to use prediction. By matching predicted resource availability windows [7] [8] with predicted job runtimes, fewer cycles are wasted.

When employing prediction based scheduling the effectiveness is largely dependent upon the quality of the predictions being made. Unfortunately, both predicting job runtimes [9] as well as predicting resource availability is a difficult problem. For this paper the job runtime prediction facet is reduced to the set of parameter sweep applications. This allows the usage of existing modeling techniques to improve prediction accuracy. A job runtime prediction technique is developed based on a number of models. These will be explicitly adapted to ensure optimal performance in the parameter sweep runtime prediction framework we refer to as GIPSy. Using GIPSy it is possible to dynamically generate accurate runtime predictions for parameter sweep jobs requiring no a-priori knowledge of the application.

To demonstrate a possible use of such a prediction framework within a grid scheduling context, GIPSy is combined with PGS [10]. PGS is a scheduling technique and grid implementation based on two constraints: the availability of job runtime estimates and resource availability estimates. Using these constraints the technique pro-actively prevents failures from occurring by distributing jobs only to resources estimated to be available for the full execution time of the job. The technique takes into account the dynamic changes that occur in the grid system instead of merely reacting to them. Resource availability estimates can be obtained depending on the type of resource used. We can distinguish two different categories, predicted availability and planned availability. Planned availability could be used in the case of workstations that are periodically available e.g. during non-work hours, or dedicated grid resources that are periodically unavailable e.g. during maintenance. Predicted availability presents a more complex problem. Predictions have to be made by monitoring the grid system. Methods have already been developed in specific areas, such as Fine-Grained Cycle
Sharing systems [11], as well as more general approaches suitable for both desktop and enterprise environments [7] [8]. For the implementation and testing of both the PGS grid component and the combination with PGS, the CoBRA grid system is used.

The rest of the paper is organized as follows. First an overview is given of the GIPSy framework in section II. Details about the components used in the tests are presented. An introduction to PGS is given in section III, describing the scheduling technique and how it uses GIPSy to achieve its goal. Section IV contains a summary of the test setup used and an extensive description of the results obtained. Finally future work and a conclusion is given in section V.

II. GIPSy

The design idea behind GIPSy has two main goals: creating a framework for testing statistical prediction mechanisms and providing this framework as a library for use in other projects. By using traditional modeling techniques in new ways it is possible to accurately predict job runtimes in parameter sweeps. Useful predictions can be achieved both in theoretical testing scenarios as well as in more realistic environments.

A. Prediction Technique

GIPSy is based on three distinct stages (see figure 1):

1) Initial Sample Selection: No runtime information is available and a set of jobs is selected whose runtimes are likely to produce a good initial model. In the presented tests the jobs are selected using the Basic Initial Sample Selector. Samples are chosen based on the minimum and maximum values of the parameters, resulting in an initial model containing the extrema of the problem domain.

2) Model Construction: Modeling the problem domain consists of finding a function that takes the job parameters as input and returns an estimate of the runtime. The framework contains many different model types: Radial Basis functions [12], Kriging Models [13], Nearest Neighbour prediction [14] [15], Neural Networks, Support Vector Machines [16] [17] and Polynomial approach [18]. The first two were used to obtain the results in section IV and are explained in detail further down.

3) Sample Selection: The order in which new sample points are obtained is one of the most influential methods to steer the model building process. Since the order of execution is irrelevant in parameter sweeps, sample selectors are free to order the jobs depending on the requirements set for the model. The most obvious choice would be to select the jobs that minimize the overall modeling error. This means running the jobs with the largest error first which produces great results when running simulated tests. However, when the predictions are used during the run an actual parameter sweep it is often better to sacrifice some accuracy to improve current predictions. To perform the tests in section IV three different sample selectors were used Smallest Distance, Sorted and Biggest Distance.

B. Models

- Radial Basis function (RBF): The approximation function \( y(c) \) consist of a weighted sum of \( N \) radial basis functions, each with a different center \( c_i \) (the sample point) and a corresponding weight \( w_i \). The resulting function is shown in equation 1.

  \[
  y(x) = \sum_{i=1}^{N} w_i \phi(\|x - c_i\|)
  \]

  Many different kernel functions can be used, in the presented benchmarks a Multiquadric kernel is chosen: \( \phi(r) = \sqrt{r^2 + s^2} \).

- Kriging: The Kriging technique is similar to RBF; the main difference being the location of the kernel function \( \phi \). Instead of building a single function around each sample point \( i \), a function is built around each dimension \( j \) of samplepoint \( i \), resulting in equation 2.

  \[
  y(x) = \sum_{i=1}^{N} \left( \prod_{j=0}^{dsm} \phi(x_j - c_{ij}) \right)
  \]

  For our tests we will be using an Inverse Multiquadric kernel function.

- Nearest Neighbour: An interpolation technique based on a value composer and the k nearest neighbours of a sample point. The value composer can be any number of functions, for the tests a local predictor was chosen based on the Kriging technique with an Inverse Multiquadric kernel, 20 samples are chosen to build each model.

C. Sample Selectors

- Smallest Distance: Intuitively, a model is most accurate when selecting points that are closest to the already calculated sample points. For each sample point the 4 closest neighbours are chosen, these points are then sorted based on overall distance from their sample point. All inputdata is normalized before the distance calculation, this is done to remove the influence of any possible scale difference between the dimensions. This technique results in better predictions early on, the downside is a relatively larger overall error across the entire data set.

- Sorted: With the previous technique we tried to improve the prediction results hopefully resulting in better scheduling. Here we use the prediction results and try to influence and improve the scheduling itself. Predictions are assumed to be reliable and new samples are chosen based on their predicted runtime. By choosing the longest runtimes first the problem round-trip-time is reduced as gaps can be filled easier by smaller jobs.

- Biggest Distance: The exact opposite of the Smallest Distance technique. By selecting new sample points as far away as possible from the known sample points the model
is improved much faster. This results in worse predictions for the first samples but a smaller overall error.

III. PGS

The goal PGS tries to achieve is the minimization of round-trip-time and optimization of throughput. It does so by anticipating changes in resource availability through the use of predictions. In this section, a short overview of the different steps necessary for scheduling is given. An overview of the way GIPSy is used to enhance the scheduler can be found in Figure 1.

A. Overview

The PGS implementation can be divided into three processes:
- The actual scheduling process, allocating jobs onto resources. Described in detail further in this section.
- A process which continuously submits jobs from the local resource queues to the actual resources.
- A process which interfaces with the GIPSy modelbuilder to acquire up-to-date runtime predictions.

New models, based on the continually growing amount of job runtime data, need to become available as soon as possible. Since they are so important for the performance of the scheduling, it is necessary to separate this process and update the runtimes during the other scheduling steps. The actual scheduling process is designed so this does not result in concurrency problems or significant slowdowns.

The scheduling process itself can be broken down into the following steps:

1) **Job Submission**: All jobs are submitted to the scheduler one by one. During submission jobs are randomly distributed over the available resources. When all jobs have been submitted an initial sample selection is performed. The corresponding jobs are placed in the front of the queue.

2) **Gathering Information**: Lists are made containing currently available resources and all jobs that still require scheduling, this includes jobs submitted to resources that have not yet finished successfully. Estimated uptimes are requested for the available resources. Using these estimates and the list of currently scheduled and running jobs the remaining resource uptime is calculated.

3) **Find First Job**: A first job is found for each resource with no jobs on the queue. This prevents CPU time from going to waste while resources wait for a job to be placed on the queue during the next step. Resources with the least amount of time available are filled first, this is a resources centric step.

4) **Filling Queues**: Resource queues are filled with jobs as long as there is still room left in their queues and while the global job queue contains viable candidates. This step is job centric and tries to minimize total round trip time by finding the best possible job distribution for the current resource pool.

5) **Job Balancing**: If necessary, the jobs on the resource queues are balanced out. This means one of two things: either resources no longer have enough time left to run all queued jobs or there are resources with multiple jobs while others have none. The first reason can be attributed to a running job taking longer than predicted, new predictions requiring more time for queued jobs or a lower resource availability estimate. The second case occurs whenever it is possible to donate jobs from busy resource queues to empty resources.

6) **Return Results**: Upon successful completion each job returns the job parameters, the predicted runtime and the actual runtime. This information is then used in the next model building iteration. Models are built continuously using a snapshot of the currently available information.

Prediction error is handled by an algorithm previous prediction and runtime information. It extrapolates a reasonable but sufficiently long runtime, ensuring consistency whenever a job fails due to a lack of processing time. More details on these and other components discussed here can be found in [10] and...
This section evaluates prediction based scheduling by simulation and real time scheduling and compares those two approaches. The simulation is based on the GIPSy testing technique described in [19]. The real time scheduling is evaluated by comparing roundtrip times using the PGS [10]. The main idea is to demonstrate that both approaches result in the same conclusions in order to prove the value of the less timeconsuming GIPSY simulation technique.

The evaluation of GIPSy and PGS is based on three different distributions described in table I. The first one implements the quantum physics problem described in [20]. The other two distributions simulate a workload represented by sleepjobs. The simulated workloads are generated by the testgenerator presented in [6]. More details about those distributions are described in table I.

![Image](image1.png)

In order to get a detailed reflection of the different techniques each possible modeling and sampleselection combination is tested. Because the scope of these tests is too large to fit in this paper, only the useful results are shown.

The GIPSy simulation selects one samplepoint at a time and creates a new model using this extra samplepoint. This process continues until every samplepoint is selected and used in the model. The results of these simulations are shown in figures 2 and 3. The Y-axis represents the relative error of a samplepoint on selection(res). De X-axis represents every samplepoint sorted by its relative error.

The results in figure 2 show three modeling techniques suitable for predicting runtimes:

- **KringingIMQ**: Kriging model with an Inverse Multi-Quadric kernel.
- **RMFMQ**: Radial Base Function with a MultiQuadric kernel.
- **NNKringingIMQ**: Nearest Neightbour model using KringingIMQ models.

Each of those models is relatively stable and has an acceptable performance in every tested situation.

Figure 3 shows the impact of the different sampleselectors on the robustness and performance of the predicton technique. Using the smallest distance sample selector clearly leads to the best results.

To conclude the GIPSy simulation tests we can state that optimal predictions are obtained when using a RBFMQ or a KringingIMQ or a NNKringingIMQ model in combination with the SD dataselector.

The second part of this evaluation focuses on real time scheduling by simulation and real time scheduling and compares those two approaches. The simulation is based on the GIPSy testing technique described in [19]. The real time scheduling is evaluated by comparing roundtrip times using the PGS [10]. The main idea is to demonstrate that both approaches result in the same conclusions in order to prove the value of the less timeconsuming GIPSY simulation technique.

The evaluation of GIPSy and PGS is based on three different distributions described in table I. The first one implements the quantum physics problem described in [20]. The other two distributions simulate a workload represented by sleepjobs. The simulated workloads are generated by the testgenerator presented in [6]. More details about those distributions are described in table I.

![Image](image2.png)

In order to get a detailed reflection of the different techniques each possible modeling and sampleselection combination is tested. Because the scope of these tests is too large to fit in this paper, only the useful results are shown.

The GIPSy simulation selects one samplepoint at a time and creates a new model using this extra samplepoint. This process continues until every samplepoint is selected and used in the model. The results of these simulations are shown in figures 2 and 3. The Y-axis represents the relative error of a samplepoint on selection(res). De X-axis represents every samplepoint sorted by its relative error.

The results in figure 2 show three modeling techniques suitable for predicting runtimes:

- **KringingIMQ**: Kriging model with an Inverse Multi-Quadric kernel.
- **RMFMQ**: Radial Base Function with a MultiQuadric kernel.
- **NNKringingIMQ**: Nearest Neightbour model using KringingIMQ models.

Each of those models is relatively stable and has an acceptable performance in every tested situation.

Figure 3 shows the impact of the different sampleselectors on the robustness and performance of the predicton technique. Using the smallest distance sample selector clearly leads to the best results.

To conclude the GIPSy simulation tests we can state that optimal predictions are obtained when using a RBFMQ or a KringingIMQ or a NNKringingIMQ model in combination with the SD dataselector.

The second part of this evaluation focuses on real time scheduling by simulation and real time scheduling and compares those two approaches. The simulation is based on the GIPSy testing technique described in [19]. The real time scheduling is evaluated by comparing roundtrip times using the PGS [10]. The main idea is to demonstrate that both approaches result in the same conclusions in order to prove the value of the less timeconsuming GIPSY simulation technique.

The evaluation of GIPSy and PGS is based on three different distributions described in table I. The first one implements the quantum physics problem described in [20]. The other two distributions simulate a workload represented by sleepjobs. The simulated workloads are generated by the testgenerator presented in [6]. More details about those distributions are described in table I.

### Table I

<table>
<thead>
<tr>
<th>Testset</th>
<th>Aantal taken (T)</th>
<th>Sequentiële runtime (T × t_m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QF Test</td>
<td>56/70</td>
<td>± 120000 s</td>
</tr>
<tr>
<td>Test 1</td>
<td>4000</td>
<td>23907.203 s</td>
</tr>
<tr>
<td>Test 2</td>
<td>2457</td>
<td>100769.1452 s</td>
</tr>
</tbody>
</table>

![Image](image3.png)
scheduling tests. It proves the relevance of prediction based scheduling on desktop grids and compares the results with those of the GIPSy simulation. Figure 4 presents the roundtrip times of the distributions using PGS in combination with the different GIPSy prediction techniques.

It is important to notice that first-come, first-served scheduling of a distributed problem on a desktop grid can lead to infinite roundtrip times. This because the chance that a job wont fit into the availability window of a selected resource remains in each rescheduling step.

The results in figure 4 lead to two conclusions:
1) Performance is optimal when using the RBFMQ model as predicted by the GIPSy tests.
2) The performance is optimal when using the Biggest Distance or Sorted data selector.
The latter conclusion contradicts the conclusions of the GIPSy tests. This contradiction is caused by the different optimization goals. The GIPSy tests try to minimize the mean prediction error of each samplepoint at execution. PGS tries to minimize the roundtrip time. Tests prove the impact of errors increases at the end of the distribution. As the Smallest distance data selector realizes a minimization of the mean error and the Biggest distance data selector minimizes the errors at the end of the distribution, the latter performs better in the PGS tests. Figure 5 shows this optimization difference by plotting the subsequent samplepoint errors chronologically. The error of the Biggest distance sample selector in figure 5(a) is minimized at the end. The smallest distance sample selector in figure 5(b) enforces a minimization of the mean error.

Future work includes optimizations of the used prediction techniques (ex. Combining the sorted and biggest distance sample selector, ...), integrating the scheduling techniques into different distribution platforms and applying the runtime prediction techniques to other domains (Grid Economics, distributed metamodelling, ...).

V. Conclusion & Future work

This paper demonstrates the capabilities of a PGS applied to desktop grids by minimizing the roundtrip time of existing distributed problems using GIPSy runtime predictions. It also shows that the GIPSy simulation principles are correct and indicates the importance of a correct optimization goal definition in order to retrieve the correct GIPSy results.

REFERENCES