

Runtime Prediction based Grid Scheduling of Parameter Sweep Jobs

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

Abstract—This paper examines the problem of predicting job runtimes by exploiting the properties of parameter sweeps. A new parameter sweep prediction framework GIPSy (*Grid Information Prediction System*) is introduced. Predictions are made based on prior runtime information and the parameters used to configure each job. The main objective is providing a tool combining development, simulation and application of prediction models within one framework. The different kinds of available sample selectors and models are discussed in detail. Results are presented for a quantum physics problem. A previously introduced scheduling technique and the implementation called PGS (*Prediction based Grid Scheduling*) is improved and presented in combination with GIPSy to obtain a real-world grid implementation that optimizes the distribution of parameter sweeps.

I. INTRODUCTION

The demand for computational resources within grid systems has grown significantly during the last years. This has lead to situations where traditional dedicated grids can no longer satisfy that demand. In an effort to increase the amount of available resources one can turn to the concept of *CPU harvesting* on idle resources. Organizations and private users all over the world have large amounts of user-controlled resources (e.g. workstations, personal computers) that often go unused over long periods of time. These resources could be made available to heavily used grid systems without the need to invest in extra equipment and infrastructure. These new platforms are called Desktop grids. They can be combined with high performance clusters into one *heterogeneous grid*, allowing user-controlled resources and clusters to co-exist within a single grid. Examples of such systems include BOINC [1](cfr. Seti@home [2]), CoBRA [3] [4] and Condor [5] [6].

Despite of their success there are some drawbacks that need to be taken into consideration. As the resources are not owned by a single instance, availability is dependent on the requirements of the owner. This means resources can join or leave the grid at any given moment. When a resource is no longer available its running jobs need to be restarted elsewhere, one does not want to wait for the resource to rejoin the grid as it is uncertain if or when the resource will return. To reduce the need for rescheduling, and thus the loss of effective CPU cycles, job runtime prediction can be used to match predicted *availability windows* [7] [8] on volatile machines.

Job runtime prediction, however, is a challenging problem [9]. Reducing this problem to the set of parameter sweep

applications offers the opportunity to use existing modeling techniques to improve prediction accuracy. These techniques combine prior job runtime information with the parameters used to configure the jobs to create a runtime model. In this paper we will develop a job runtime prediction technique based on a number of models. These will be explicitly adapted to ensure optimal performance of the parameter sweep runtime prediction framework we will refer to as GIPSy (*Grid Information Prediction System*).

After introducing GIPSy we will combine it with an improved version of PGS [10], a scheduling technique and 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 available for the full execution time of the job. No CPU cycles are wasted on jobs that will be unable to complete. GIPSy allows the initial runtime predictions and subsequent updates to be made dynamically. The 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 proposed combination with PGS, the CoBRA grid system is used.

The rest of the paper is organized as follows. In section II the currently available runtime prediction techniques are discussed. Section III gives an overview of the different modeling techniques used for prediction. In section IV we introduce a novel statistical prediction technique and we describe the software framework built to test and use the prediction techniques. The quality of the proposed framework is tested by predicting the runtimes of a distributed Quantum Physics problem. Improvements to PGS can be found in section V.

The integration of GIPSy within PGS is discussed in section VI. A conclusion and overview of future work can be found in the final section.

II. RELATED WORK

There are three major approaches for estimating runtimes: *code analysis*, *analytical benchmarking/code profiling* and *statistical prediction*.

The *code analysis* approach counts the number of instructions to be executed for a given input [12]. Since this is a low-level approach it is typically limited to a specific code type and a limited set of architectures.

Analytical benchmarking/code profiling is based on a set of primitive routines. A code profiling tool divides each task in a combination of this set of primitive routines. The runtime is predicted by combining the runtimes retrieved from benchmarking the set of primitive routines [13] [14] [15] [16] [17] [18] [9]. Exhaustive profiling and benchmarking results in very accurate estimates but are not suitable for parameter sweeps as the input parameters do not influence the runtime predictions.

Statistical prediction is an approach based on the statistical properties of the runtimes in past observations. Prediction techniques following the statistical benchmarking approach can be categorized by two characteristics:

- *Sample set*: Some techniques use runtime samples obtained prior to the current parameter sweep run (Ex. [19] [20]). Other use only the samples of the current run [9].
- *Prediction technique*: Many techniques are available, only a partial overview is given. One group of techniques creates a model of the runtimes which can be queried to obtain the required runtime of a certain job (Ex. [21]). Another group creates categories of similar jobs and obtains the runtime prediction by calculating mean values of those groups. These techniques are based on a definition of similarity and an algorithm to detect this similarity [19] [22]. Yet another group uses finite state machines to predict the runtime [23].

Our approach will be situated in the statistical prediction category.

III. USED MODELS

Given that a parameter sweep involves a set of N parameters $p = (p_1, \dots, p_n)$, we can introduce the multi-dimensional parameter space P . Each parameter sweep job $J(p_1, \dots, p_n)$ corresponds to a point (p_1, \dots, p_n) in space P . The prediction problem can be reformulated as the problem of modeling the runtime function $Exec : P \mapsto \mathbb{R}_+ : (p_1, \dots, p_n) \mapsto Exec(p_1, \dots, p_n)$. This section describes seven model types:

Polynomial approach: This technique constructs multi-dimensional polynomial functions of degree n based on the available sample points using a standard least squares [24] approximation.

Radial Basis functions (RBF) [25]: Function approximation using radial basis functions resembles the polynomial approach in the sense that it uses the least squares approximation technique to construct a model defined by equation (1).

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

The approximating function $y(x)$ is represented as a sum of N radial basis functions, each associated with a different center c_i , and weighted by a coefficient w_i . We will consider three specific kernel functions (s is a scaling factor):

- Gaussian: $\phi(r) = e\left(\frac{-r^2}{s^2}\right)$
- Multiquadric: $\phi(r) = \sqrt{r^2 + s^2}$
- Inverse (Reciprocal) Multiquadric: $\phi(r) = \frac{1}{\sqrt{r^2 + s^2}}$
- Epanechnik [26] [27]: $\phi(r) = \frac{3}{4}\left(1 - \left(\frac{r}{s}\right)^2\right)$

New kernel functions can be easily added later.

Kriging Models [28]: The Kriging and RBF technique are identical except for the form of the model. It is based on the same kernel functions ϕ and also uses the least squares approximation technique to construct its model. A Kriging model has the following form:

$$y(x) = \sum_{i=1}^N \left(w_i \prod_{j=0}^{dim} \phi(x_j - c_{ij}) \right) \quad (2)$$

Neural Networks: The neural networks to be considered in this paper are characterized by the following properties:

- Layers: Number of hidden layers in the network
- Layer types: Types of the threshold functions used in each layer (sigmoid, linear, tanh, logarithmic).
- Layer size: Number of perceptrons in the layer.
- Learningrate: Effect of the error on the new weights during each learning cycle. Expressed by η in equation 3.

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \eta \delta_j^{(l)}(n) y_i^{(l-1)}(n) + \alpha [w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)] \quad (3)$$

- Cycles: Amount of training cycles used to construct the final model.
- Momentum: Parameter α used to prevent sudden weight changes. A large α results in a large prevention, $\alpha = 0$ means no prevention (See equation 3).
- Learner: Used learning algorithms (standard backpropagation or resilient backpropagation).

A model is created by training the configured neural network in the specified amount of cycles using the defined learning algorithm.

Two different implementations of this technique are used. One based on the Matlab Neural Network Toolbox [29] and one using Joone (Java Object Oriented Neural Engine) [30].

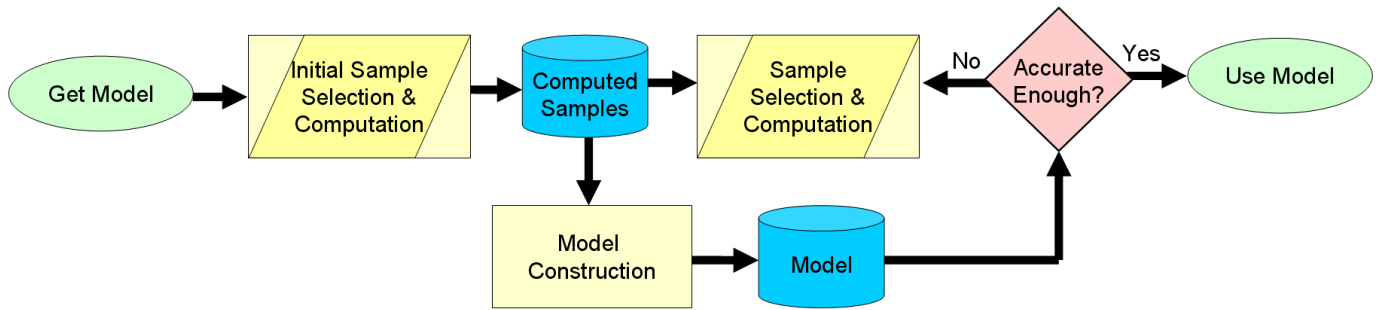


Fig. 1. Three prediction stages.

Support Vector Machines: This technique [31] [32] is based on training an entity classification using a hyperplane. Kernel functions like those explained in the RBF section are used to map the original non-linear observations into a higher-dimensional space, where the linear classifier can subsequently be used to classify this original non-linear data. The implementation that will be used here is based on the Java library *LibSVM* [33].

Nearest Neighbor prediction: This technique [34] [35] is an interpolation prediction technique which calculates the runtime of a job based on a function F (the *valuecomposer*) and the k nearest neighbors of this job in the multi-dimensional¹ sample space. The parameter k as well as the function F are predefined by the user. In the current study we will consider four types of value composers: (1) calculates the mean value of the k points (standard interpolation), (2) calculates the standard weighted mean value using a linear weight function on the distances of the samples in the multi-dimensional space, (3) calculates the standard weighted mean value using a non-linear weight function (e.g. *Epanechnikov Kernel*) on the distances (4) uses a local predictor based on one of the previous model types.

Technique of Iverson et al.: This technique [21] is a special case of the Nearest Neighbor prediction mechanism.

IV. PREDICTION FRAMEWORK

A. The technique

This section describes a new statistical parameter sweep prediction methodology called GIPSY and introduces two new model optimization mechanisms: (1) *Model Splitting* and (2) *Outlier reduction*. As no a priori assumptions can be made about similarities in the runtime behavior of independent parameter sweeps, we will only consider data from the current run. This means that the similarity based job categorizing technique and the finite state machine technique previously described will not be used.

GIPSY is based on three stages (see Fig. 1):

¹The dimension is described by the number of input parameters and the position of each job in this space is defined by its input parameter values.

- **Initial sample selection:** In this stage an initial sample set is selected, a set of jobs scheduled without runtime predictions. This is necessary to gather the data required to start up the modeling phase. We will consider three initial sample selection methods:

- *RandomInitialDataSelector*: Selects random samples to create the start grid.
- *BasicInitialDataSelector*: Selects samples containing a minimum or maximum value for one of the parameters.
- *ZeroSelector*: Starts the modeling phase without selecting initial sample points.

- **Model Construction:** The accuracy of the prediction results strongly depends on the chosen modeling method. A number of model types are available within the prediction framework, these were described in section III.

- **Sample selection:** The order of execution in a parameter sweep is irrelevant. Hence this stage will attempt to schedule the jobs in an order that minimizes the modeling error. However, this often translates in running the jobs with the worst prediction first. Since this is not always desirable, one can choose to minimize the overall modeling error during the entire run or alternatively sacrifice some accuracy to achieve better predictions for the current samples. Model convergence will depend on the specific sample selection technique chosen. This paper will consider five implementations, each of which can be used for all models:

- *RandomDataSelector*: Selects random samples from the remaining parameter space.
- *MultiPredictorDataSelector*: Selects samples based on the difference between predefined models in each of the remaining sample points.
- *GriddedDataSelector*: Selects samples based on a predefined sample grid.
- *LESDDDataSelector*: Selects samples based on the combination of the difference between two predefined models in each of the remaining sample points and on sample density.
- *CompleteDataSelector*: Similar to the *MultiPredictorDataSelector*. It works in cycles and selects in each

cycle the sample with the largest difference to the other models. At the end of each cycle the models are updated with the predicted sample value.

- *NNDataSelector*: Select the samples closest to the verified sample points.

B. Model splitting

The impact of some parameters on the runtime cannot be fit in a model as their influence on the result is random. Those parameters are detected and for each value of those parameters a separate model is constructed.

C. Outliers reduction technique:

When building a model, the data of some samples will be critical to construction. These samples are called the critical sample points and can also be defined as the points where the model function derivative reaches zero. This means that these critical points contain the maxima, the minima and the inflection points of the model function. As our technique is supposed to gather the info of those critical sample points in an early stage of the modelbuilding process all values predicted by the model should, at some point, be between the maximum and minimum value of those sample points. This technique assumes that this requirement is fulfilled from the beginning² and removes all possible outliers by returning the minimum/maximum value when the predicted value is smaller/larger than the minimal/maximal sample value. This error reduction of the incorrect sample point predictions improves the accuracy of the model.

D. Implementation

The goal for GIPSy is to provide a framework capable of testing statistical prediction mechanisms, and providing a library to use those prediction mechanisms in an independent software project. Every component in the framework is defined by a configfile describing the component specific configuration parameters, the classes needed³ to load the specific component and the location of the Jar files containing the implementation of the component. New component implementations can simply be *plugged-in* by using these configfiles. The whole range of model implementations is Java based using the “JAMA” matrix package [36] and a Jini [37] [38] based access mechanism to the Matlab [39] toolbox.

E. Testing and results

In this section the framework is tested by predicting the runtime of a distributed quantum physics problem [40]. These predictions are performed for each of the initial sample selectors, for each of the sample selectors and for each type of model. Tests show that a *well spread initial sample selection*

²This false assumption only results in incorrect predictions in the first cycles of the model construction cycle where the error rate is still too large to be affected by these incorrect predictions.

³Prediction components need a prediction specific factory and the prediction class to be loaded (Ex. PolynomialFactory & Polynomial).

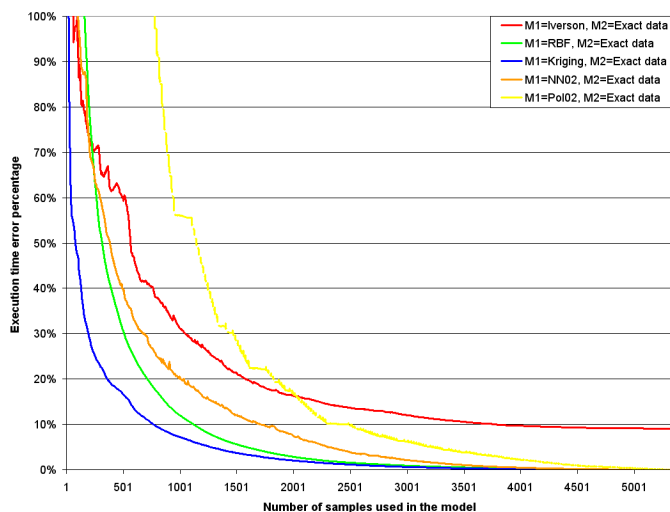


Fig. 2. Graph constructed using an exact data based sample selector.

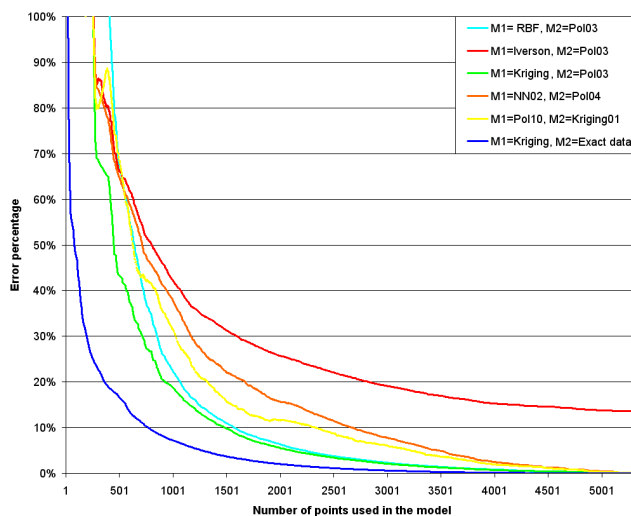


Fig. 3. Graph constructed using a sample selector based on an additional model. Kriging compared with exact data is added for reference.

in combination with *error based sample selection*, *outlier reduction* and *model splitting* give the best overall results using any model. The best results were obtained using the *BiggestErrorDataSelector*⁴ and *BasicInitialDataSelector* classes. These are used in combination with the different models to produce the presented benchmarks.

The experiments performed add one sample point to the available data in each iteration (X-axis in figure 2 and 3). The Y-axis shows the error percentage of the prediction ($error = \frac{|\bar{t}-t|}{t} * 100$ with \bar{t} the predicted runtime and t the exact runtime).

The graph on figure 2 shows the results using the exact runtimes to select new sample points (the sample with the largest prediction error $e = \bar{t} - t$ is selected). The plot shows that Kriging prediction is the optimal technique when selecting

⁴A MultiPredictorDataSelector that selects the sample points with the largest prediction difference between the models.

Scenario	0	1	2	3
PGS Improved (Minutes)	63.54	65.05	65.5	75.8
PGS Old (Minutes)	63.54	65.54	68.46	87.88
FCFS (Minutes)	63.21	74.26	87.3	121.83
PGS Improved STDEV	0.19	0.15	0.22	1.17
PGS Old STDEV	0.19	0.39	0.73	2.03
FCFS STDEV	0.12	0.63	1.68	2.25
% Difference Old & Improved	0%	0.74%	4.33%	13.74%

TABLE I
COMPARISON BETWEEN PGS OLD, IMPROVED AND FCFS

the sample points in the most ideal way. The Kriging results are added to figure 3 as a reference. The graph in figure 3 demonstrates the results when models are built using sample selection based on a second model (M2) to calculate the prediction error. For each prediction model M1 the results are shown for the most ideal model M2 used in the sample selector. The results show that for this data Kriging and RBF are the most accurate prediction models in combinations with a low degree Polynomial. Some other test indicate that Kriging and RBF tend to be stable in other runtime predictions as well but additional tests are needed.

V. PGS IMPROVEMENTS

PGS is a dynamic fault-aware scheduling mechanism built on top of the CoBRA framework. It uses job runtime predictions and resource availability predictions to improve performance of *Bag-of-Tasks* (BoT) applications [41]. For the purpose of this implementation we will narrow the field of BoT applications down to parameter sweeps. The techniques used in PGS and the implementation details can be found in [10]. The most recent version of the PGS scheduler includes some small changes, intended to enhance performance, that can be viewed separate from the changes made to accommodate the prediction errors introduced by using the GIPSy framework.

A. Load Balancing

In addition to some smaller changes the most significant improvement was achieved through the implementation of a load balancing system. The intended purpose is twofold. First, it removes jobs from busy resource queues when other resources are idle and have enough uptime left to successfully complete one of the queued jobs. Second, it removes jobs from resource queues when the total remaining uptime is insufficient to complete all queued jobs.

B. Testing and results

To determine the impact of these changes the tests conducted in [10] are repeated. Figure 4 shows that using the improved PGS a speedup of up to 13.74% can be achieved when compared to the previous version. More detailed results can be found in table I. When looking at the standard deviation, the results also show that the round trip times are not only shorter but also more consistent when using load balancing.

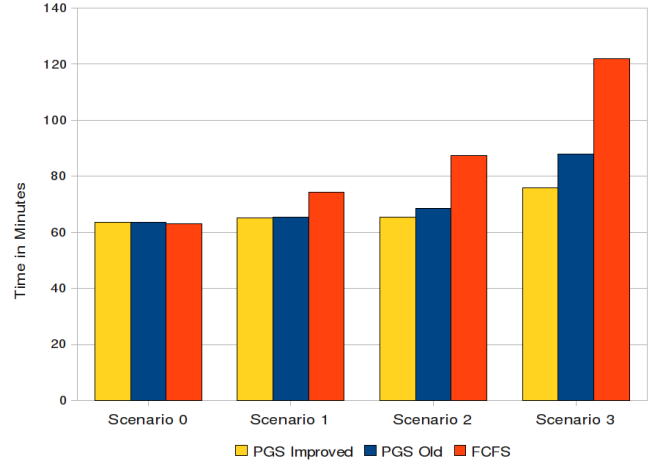


Fig. 4. Test results using the improved scheduler

VI. COMBINATION WITH PGS

In this section we describe the integration of GIPSy within the CoBRA grid and the way it interacts with PGS to improve grid performance.

A. Implementation

The current GIPSy framework is adapted to work within the CoBRA [4] middleware by turning it into a pluglet that can be used similar to other different resources within a running CoBRA grid. In its CoBRA pluglet form it can be described as a *Pluggable Multipurpose Prediction generator*. When a user submits a parameter sweep to the grid the desired prediction pluglet must be specified, multiple prediction pluglets may be available each configured in a different way. When the desired pluglet is found a description of all jobs, consisting of the parameters used to configure the jobs, is sent to the pluglet so it can start initial sample selection. When all jobs are submitted these samples are scheduled first. When the first jobs finish and runtime data becomes available a model is built and new samples are selected. The obtained model is used to add a predicted runtime to each job that is still available to the scheduler. New models are built on a constant bases taking into account the new runtime data.

B. Handling prediction error

When a job is executing on a resource its actual runtime is recorded, when the job exceeds the runtime prediction a new dynamic prediction d_p is made. The initial value for the dynamic prediction is the initial predicted runtime. The new dynamic prediction is based on three factors: average amount by which previous predictions fell short e_a , the average percentage by which they fell short e_f and the longest recorded resource uptime l_u . A new dynamic runtime is calculated as follows in equation 4.

$$d_p = \min(d_p * e_f, d_p + a_r, l_u) \quad (4)$$

When a jobs fails, due to prediction errors in either job runtime or resource availability, it re-enters the scheduling

queue. Subsequent runtime prediction updates can not result in a shorter prediction than its previous dynamic runtime prediction.

VII. CONCLUSION & FUTURE WORK

This paper introduces the GIPSy prediction framework as well as an update to the PGS scheduler. Each is validated separately and the integration details of GIPSy within PGS are presented. Future work will consist of further improvements in both the combination of PGS and GIPSy as well as the individual components. Thorough benchmarks are needed to further validate the presented approach and see how well these results translate into a real-world grid configuration.

REFERENCES

- [1] BOINC, "Available at <http://boinc.berkeley.edu/>." [Online]. Available: <http://boinc.berkeley.edu/>
- [2] Seti@Home, "Available at <http://setiathome.ssl.berkeley.edu/>." [Online]. Available: <http://setiathome.ssl.berkeley.edu/>
- [3] P. Hellinckx, G. Stuer, W. Hendrickx, F. Arickx, and J. Broeckhove, "Grid-user driven grid research, the CoBRA grid," in *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06)*. Washington, DC, USA: IEEE Computer Society, 2006, p. 49.
- [4] P. Hellinckx, F. Arickx, J. Broeckhove, and G. Stuer, "The CoBRA grid: a highly configurable lightweight grid." *International Journal of Web and Grid Services*, vol. 3, pp. 267–286(20), 20 August 2007.
- [5] Condor, "Available at <http://www.cs.wisc.edu/condor/>."
- [6] P. Hellinckx, G. Stuer, D. Dewolfs, F. Arickx, J. Broeckhove, and T. Dhaene, "Dynamic problem-independent metacomputing characterization applied to the Condor system," in *Proceedings ESM2003 p.262-269*, 2003.
- [7] R. Wolski, "Dynamically forecasting network performance using the network weather service," *Cluster Computing*, vol. 1, no. 1, pp. 119–132, 1998. [Online]. Available: citeseer.ist.psu.edu/wolski98dynamically.html
- [8] J. Brevik, D. Nurmi, and R. Wolski, "Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems," in *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 190–199.
- [9] M. A. Iverson, F. Özgüner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Trans. Computers*, vol. 48, no. 12, pp. 1374–1379, 1999.
- [10] S. Verboven, P. Hellinckx, F. Arickx, and J. Broeckhove, "Dynamic grid scheduling using job runtime requirements and variable resource availability," in *Proceedings of EuroPar 2008*, 2008.
- [11] X. Ren, S. Lee, R. Eigenmann, and S. Bagchi, "Resource availability prediction in fine-grained cycle sharing systems," in *Proceedings of the Conference on High Performance Distributed Computing*, 2006.
- [12] B. Reistad and D. K. Gifford, "Static dependent costs for estimating execution time," *SIGPLAN Lisp Pointers*, vol. VII, no. 3, pp. 65–78, 1994.
- [13] R. F. Freund, "Optimal selection theory for superconcurrency," in *Supercomputing '89: Proceedings of the 1989 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 1989, pp. 699–703.
- [14] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L. Wang, "Heterogeneous supercomputing: problems and issues," in *Proc. of the 1992 Workshop on Heterogeneous Processing*, 1992.
- [15] —, "Heterogeneous computing: Challenges and opportunities," *Computer*, vol. 26, no. 6, pp. 18–27, 1993.
- [16] D. Pease, A. Ghafoor, I. Ahmad, D. L. Andrews, K. Foudil-Bey, T. E. Karpinski, M. A. Mikki, and M. Zerrouki, "PAWS: A performance evaluation tool for parallel computing systems," *Computer*, vol. 24, no. 1, pp. 18–29, 1991.
- [17] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of execution times on heterogeneous supercomputer architectures," in *ICPP '93: Proceedings of the 1993 International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1993, pp. 219–226.
- [18] T. Yang and A. Gerasoulis, "DSC: Scheduling parallel tasks on an unbounded number of processors," University of California at Santa Barbara, Tech. Rep. TRCS94-12, 20, 1994. [Online]. Available: citeseer.ist.psu.edu/125354.html
- [19] W. Smith, I. T. Foster, and V. E. Taylor, "Predicting application run times using historical information," in *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1998, pp. 122–142.
- [20] F. Nadeem, M. M. Yousaf, R. Prodan, and T. Fahringer, "Soft benchmarks-based application performance prediction using a minimum training set," in *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2006, p. 71.
- [21] M. A. Iverson, F. Zghe, and G. J. Follen, "Run-time statistical estimation of task execution times for heterogeneous distributed computing," *HPDC*, vol. 1, pp. 263–, 1996.
- [22] A. Ali, A. Anjum, J. Bunn, R. Cavanaugh, F. van Lingen, R. McClatchey, M. A. Mehmood, H. Newman, C. Steenberg, M. Thomas, and I. Willers, "Predicting the resource requirements of a job submission." [Online]. Available: citeseer.ist.psu.edu/747807.html
- [23] M. V. Devarakonda and R. K. Iyer, "Predictability of process resource usage: A measurement-based study on unix," *IEEE Trans. Softw. Eng.*, vol. 15, no. 12, pp. 1579–1586, 1989.
- [24] Å. Björck, *Numerical Methods for Least Squares Problems*. Philadelphia: SIAM, 1996.
- [25] M. D. Buhmann and M. D. Buhmann, *Radial Basis Functions*. New York, NY, USA: Cambridge University Press, 2003.
- [26] V. A. Epanechnikov, "Non-parametric estimation of a multivariate probability density," *Theory of Probability and its Applications*, vol. 14, no. 1, pp. 153–158, 1969. [Online]. Available: <http://link.aip.org/link/?TPR/14/153/1>
- [27] R. L. Eubank, *Spline Smoothing and Nonparametric Regression*. Dekker, 1988.
- [28] M. D. Morris, "The design and analysis of computer experiments. thomas j. santner , brian j. williams , and william i. notz," *Journal of the American Statistical Association*, vol. 99, pp. 1203–1204, December 2004, available at <http://ideas.repec.org/a/bes/jnlasa/v99y2004p1203-1204.html>.
- [29] MNNT, "Available at <http://www.mathworks.com/products/neuralnet/>" [Online]. Available: <http://www.mathworks.com/products/neuralnet/>
- [30] Joone, "Available at <http://www.jooneworld.com/>." [Online]. Available: <http://www.jooneworld.com/>
- [31] A. M. Andrew, "An introduction to support vector machines and other kernel-based learning methods," *Robotica*, vol. 18, no. 6, pp. 687–689, 2000.
- [32] J. A. K. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. KU Leuven, 2002.
- [33] C.-C. Chang and C.-J. Lin, "LIBSVM – a library for support vector machines." [Online]. Available: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- [34] P. E. Hart, "The condensed nearest neighbour rule," *IEEE Transaction Information Theory*, 1968.
- [35] J. Friedman, F. Baskettand, and L. Sustek, "An algorithm for finding nearest neighbors," *IEEE Transaction on computer*, vol. 24, pp. 1000–1006, 1975.
- [36] JAMA, "Available at: <http://math.nist.gov/javanumerics/jama/>."
- [37] W. K. Edwards, *Core Jini*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.
- [38] Jini, "Available at <http://www.jini.org/>."
- [39] MATLAB, "Available at <http://www.mathworks.com/>."
- [40] J. Broeckhove, F. Arickx, P. Hellinckx, V. Vasilevsky, and A. Nesterov, "The 5h resonance structure studied with a three-cluster j-matrix model," *Journal of Physics G Nuclear Physics*, vol. 34, pp. 1955–1970, Sep. 2007.
- [41] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve, F. A. B. Silva, C. O. Barros, and C. Silveira, "Running bag-of-tasks applications on computational grids: The MyGrid approach," *ICPP*, vol. 00, p. 407, 2003.