Predictive Application-Performance Modeling in a Computational Grid Environment

Nirav H. Kapadia José A. B. Fortes Carla E. Brodley

School of Electrical and Computer Engineering Purdue University West Lafayette, IN 47907-1285 {kapadia, fortes, brodley}@ecn.purdue.edu

Abstract

This paper describes and evaluates the application of three local learning algorithms — nearest-neighbor, weighted-average, and locally-weighted polynomial regression — for the prediction of run-specific resourceusage on the basis of run-time input parameters supplied to tools. A two-level knowledge base allows the learning algorithms to track short-term fluctuations in the performance of computing systems, and the use of instance editing techniques improves the scalability of the performance-modeling system. The learning algorithms assist PUNCH, a network-computing system at Purdue University, in emulating an ideal user in terms of its resource management and usage policies.

1. Introduction

It is now recognized that the heterogeneous nature of the network-computing environment cannot be effectively exploited without some form of adaptive or demand-driven resource management (e.g., [10, 11, 12, 14, 18, 27]). A demand-driven resource management system can be characterized by its ability to make automatic cost/performance tradeoff decisions at run-time. Such decisions require that the infrastructure be able to decide how (which implementation — e.g., sequential versus parallel) and where (which platform) to execute a tool. This, in turn, implies an ability to *estimate* the resource requirements of any given run *before* a scheduling decision is made. This paper describes and evaluates the application of three local learning techniques for run-specific resource-usage prediction.

Local learning algorithms use available data in the region of a query to build a local model for the appropriate output [2]. For this domain, available data consists of measured resource-usage associated with previous runs of the given tool, and the query point is defined by the input parameters associated with the run for which resource-usage is to be predicted. The region around the query point is specified by way of a distance metric that measures "distances" within the space defined by tool-specific input parameters.

PUNCH is a distributed, web-accessible networkcomputing system that currently caters to about five hundred users, and provides access to forty research and commercial tools in computer architecture, parallel programming, and computational electronics. The described learning algorithms assist PUNCH [16, 20] in emulating an *ideal user* in terms of its resourcemanagement and usage policies. An ideal user is defined as one who: 1) can predict the resourcerequirements of each run that he/she initiates, 2) preferentially uses the most plentiful resources that support the requirements of the given run, and 3) voluntarily relinquishes resources to higher-priority users when necessary.

The paper is organized as follows. Section 2 outlines the characteristics of the tools and the computational grid environment in the context of resource usage prediction. Section 3 summarizes related work on the topic. Section 4 motivates the selection of instance-based learning algorithms for this domain. Sections 5 and 6 describe and evaluate the nearestneighbor, weighted average, and locally weighted regression algorithms, respectively. Finally, Section 7 presents the conclusions of this work.

2. Domain Constraints

Two sets of problems must be addressed in order to be able to predict tool- and run-specific resource-usage in a computational grid environment [18]. The first set consists of issues that are a consequence of the diversity of the tools executed on the computational grid, whereas the second set includes issues that arise due to the dynamic nature of the run-time environment.

The resources utilized by a particular run of a given tool often depend on the input to the tool. In some cases (e.g., with matrix manipulation codes), this dependence can be expressed by way of analytical expressions. In general, however, the exact relationship between the tool-input and the corresponding resourceusage is complex — making it necessary to *learn* the relationship (i.e., the *concept*).

Learning algorithms employed for resource-usage prediction must address the following three tool-related concerns: 1) the extent to which individual input parameters (i.e., *features*) affect the relationship is unknown — making it necessary to learn the relative importance of the features, 2) the range and distribution of the values of the features are not known in advance — which implies that scaling factors employed for numerical stability have to be determined on the fly, and 3) the concept to be learned often has a nondeterministic component — due to measurement noise (e.g., as with CPU time and network delays), performance variations inherent to complex computer systems (e.g., the impact of concurrently-executing applications on the effectiveness of the memory sub-system), and unobservable features (e.g., the convergence rate of many numerical algorithms depends on the eigenvalue distribution, which may not be known when the run is initiated).

The computational grid environment presents two challenges in terms of acquiring and managing knowledge for resource-usage prediction. The first challenge is a result of the real-time nature of the prediction process, which must occur after a user initiates a run, but before the run can be scheduled. This imposes an upper bound on the amount of information that can be searched and/or utilized by learning algorithms. The second challenge is a consequence of fluctuations in the availability and performance of individual nodes within large, networked computing systems. Such fluctuations can be caused by overloaded file-servers or network routers, and occur at unpredictable times. Consequently, learning algorithms employed for resourceusage prediction must be able to quickly tailor their predictions to short-term variations without being unduly affected by them in the longer term.

The application-performance modeling system for PUNCH [18, 19]: 1) employs locally weighted polynomial regression [2, 9], allowing it to work with unknown feature weights and incomplete/noisy information, 2) addresses scalability issues by way of a cache that allows it to exploit the locality of runs [18] and by selectively incorporating information into its knowledge base, and 3) tracks short-term fluctuations in performance by way of a two-level knowledge-base that differentiates between short-term memory and longterm memory. This paper describes three local learning techniques — nearest-neighbor, weighted-average, and locally weighted regression — that can be used to predict resource-usage; knowledge representation and management issues are described in [18, 19].

3. Related Work

Existing work aimed at estimating resource-usage makes use of cumulative statistical data or analytical expressions to predict run time. Statistical models are typically tool-specific, and are constructed from measured execution times of previous runs. Analytical expressions are also tool-specific, and are generally provided by the administrators or the users of the tool.

Examples of systems that utilize cumulative statistical data can be found in [1, 8, 14, 15, 26, 27, 28]. Such systems implicitly assume that a particular tool will exhibit similar resource-usage characteristics across runs — regardless of the input to the tool.¹ Although this assumption has been found to be valid for specific tools and environments (e.g., [7]), it is not true in general. For instance, multiple users concurrently executing a given tool in a grid environment with radically different input data may cause the observed resource-usage to vary rapidly from one run to the next. This problem could be partially addressed by maintaining separate statistics for each user — however, this would imply that information gleaned from one user could no longer be exploited in order to make predictions for runs initiated by other users.

The applicability of analytical expressions is restricted to the types of tools for which such expressions can be determined with relative ease (e.g., as with numerical software). As a result, this approach is of limited use in a computational grid environment. Some systems address this limitation by using analytical expressions in conjunction with other approaches — SmartNet [11, 12], for example, estimates resourceusage characteristics by employing a combination of statistical analysis and optional analytical expressions provided by users.

The design objectives of an application-performance modeling system, as stated by Berman [10], are: 1) to utilize dynamic information to represent variations in performance, 2) to produce performance predictions

¹Most of the mentioned systems can adapt to long-term changes in tool behavior by discarding "old" statistics. The system described in [26] accounts for command-line arguments supplied to the tool, when they are available.

that are timeframe-specific, and 3) to be able to adapt to a wide spectrum of potential computational environments. The PUNCH performance-modeling system utilizes local learning algorithms to *learn* the correlation between tool-specific, run-time input parameters and the corresponding run-specific resource requirements. The run-specific values of administratorspecified input parameters are automatically extracted by PUNCH from arguments and/or files supplied to the tool [21]. A two-level knowledge base [18, 19] allows the system to tailor its predictions to short-term variations in the performance of the tool and/or the computing environment. The instance-based learning algorithms utilized by the system are relatively insensitive to the structural complexity of the function to be learned [9] — consequently, they can be used for a wide range of tools.

4. Learning Algorithm Selection

Global parametric learning algorithms [24] such as neural networks attempt to establish an input-output mapping via a single function $y = f(x, \theta)$, where θ is a finite-length parameter vector. While these methods can theoretically approximate any continuous function (e.g., [13]), they may not be appropriate for all tools. For example, semiconductor device simulation tools typically allow users to simulate a device in one, two, or three dimensions. In general, different solution techniques are used for each of these cases, implying that the input-output mapping for such tools will consist of three distinct concepts. This is likely to cause problems for learning algorithms that attempt to capture concepts at a global level.

Local parametric algorithms attempt to overcome some of the problems of global parametric learning by dividing the input space into many partitions [2, 24]. Each partition *i* is approximated by an independent function $y_i = f_i(x, \theta_i)$; the functions f_i are kept as simple as possible. The problem now shifts to the selection of appropriate partitions for the learning system [25]. Non-parametric algorithms (e.g., [2, 6, 22]) address this issue by allowing the number of partitions (and consequently the number of parameters) to change dynamically. Instance-based learning (IBL) algorithms achieve this by recomputing a fixed set of parameters as a function of the query point.

IBL algorithms do not require an explicit training phase [6]. Moreover, because of their localized nature, the algorithms are relatively insensitive to the structural complexity of the function to be learned and are not affected by catastrophic interference (a condition in which previously learnt information is forgotten by an incremental learning system [24]), making them a good choice for this domain.

5. Instance-Based Learning Algorithms

This section presents three instance-based learning methods: nearest-neighbor, weighted-average (kernel regression), and locally weighted regression. The issues involved in the selection of values for the parameters that define these methods are outlined in Section 5.4. Additional details can be found in [17].

5.1. Nearest-Neighbor

K-nearest neighbor (k-NN) algorithms [2] predict the output value(s) for a given query point by using an unweighted average of the output values of the knearest instances as defined by a distance metric. Implementing a k-NN algorithm requires the specification of the following parameters: 1) the number of instances (k) to be included in the local neighborhood, and 2) an appropriate distance metric. The experiments in this paper employ a 1-NN algorithm.

5.2. Weighted Average

Weighted average algorithms [2] make their prediction on the basis of a weighted average of the output values of the k nearest instances; the weight of an instance is an inverse function of its distance from the query point. The following parameters must be specified in order to implement a weighted average algorithm: 1) the number of points to be included in the local neighborhood (kernel width), 2) an appropriate distance metric, and 3) a kernel (i.e., weighting) function. In order to quantify the benefits of using a slightly more sophisticated prediction technique (compared to the 1-NN algorithm), the experiments in this paper use a three-point weighted average algorithm.

5.3. Locally Weighted Polynomial Regression

Locally weighted regression (LWR) algorithms (e.g., [2, 3, 9]) fit a surface to nearby points, typically via a locally linear or quadratic model.² With a linear (quadratic) model, the target concept is locally approximated by a linear (quadratic) surface.

In order to clarify the ideas behind LWR, this section includes an overview of locally weighted regression. A locally linear polynomial and a dataset with one attribute are used for the purpose of the illustration; the

 $^{^2{\}rm Higher}$ order local models are generally not used because of the associated computational cost.

ideas can be extended to higher order local polynomials and datasets with multiple attributes [2, 17].

Consider linear regression analysis on data that was obtained from a function with one independent and one dependent variable (y = f(x)). Say we have Nsamples of this function corresponding to $y_i = f(x_i)$, where $1 \le i \le N$. Then, the line determined by global linear regression minimizes the sum of the squares of the errors. That is, if the line is given by

$$\hat{y} = b_0 + b_1 x, \tag{1}$$

linear regression determines b_0 and b_1 such that the error

$$\sum_{i=1}^{N} (y_i - \hat{y_i})^2$$

is minimized. Note that y_i and $\hat{y_i}$ are functions of x.

In contrast, locally weighted linear regression minimizes a weighted sum of the squares of the errors. The weights are local in the sense that they are (re)computed for each query, and the kernel function (i.e., weighting function) is chosen so as to eliminate the effects of remote data-points. The size of the local neighborhood (i.e., the region in which the weights are non-zero) is called the kernel width or bandwidth. Mathematically, locally weighted linear regression determines $b_0(x_q)$ and $b_1(x_q)$ such that the error

$$\sum_{i=1}^{N} w_{qi} (y_i - \hat{y}_i)^2 \tag{2}$$

is minimized, where x_q is the query point and w_{qi} are the query-specific weights. The coefficients of the polynomial described by Equation 1 are now functions of the query point. The weights w_{qi} are computed as

$$w_{qi} = K(d(x_q, x_i), k_w),$$

where $K(\cdot)$ is a non-negative function (the kernel function) whose value increases as $|x_q - x_i|$ decreases, $d(\cdot)$ is a distance metric, and k_w is the kernel width.

In applying locally weighted regression, four parameters must be selected: 1) the order of the local polynomial, 2) the distance metric, 3) the kernel width, and 4) the kernel function. A locally linear polynomial was used for this research because empirical evaluation [19] showed that: 1) it resulted in lower prediction errors and faster learning, and 2) it required less time than higher order models to make a prediction. The regression equations were solved by singular value decomposition [23] to ensure numerical stability.

5.4. Parameter Selection

The distance metric is common to all three algorithms. Given that the range and distribution of the values of the features are not known, a distance function that normalizes distances with respect to the query point was used. The normalization compresses dimensions in proportion to the value of the query point in the corresponding dimension, allowing the distance function to accommodate a wider range of values (for finite-precision arithmetic).³

The kernel width (i.e., bandwidth) can be fixed or variable. A fixed kernel width is of limited use because it can lead to inaccurate or undefined predictions in regions with low data-density [5]. Given the absence of a "complete" dataset in this domain, the kernel width was locally optimized [9] by recomputing it for each query. The kernel widths for the nearest-neighbor and weighted average algorithms are equal to the distance (from the query point) of the first and third nearest neighbors, respectively. For locally weighted regression, the kernel width is equal to the distance of the $2(n+1)^{th}$ nearest neighbor from the query point, where n is the length of the feature vector.

The kernel function determines the relative weights of the datapoints that fall within the kernel width. The function is required to be non-negative and have decreasing values with increasing distance [9]. A hybrid, query-dependent kernel function that maintains a constant value of one for a distance equal to that of the nearest neighbor, and is Gaussian after that was employed because it resulted in lower prediction errors when compared to the "nearest-neighbor bandwidth" described in [9]. The hybrid nature of the kernel function also ensures that at least one data-point is available to make a prediction, regardless of the datadensity in the region of the query point.

6. Empirical Evaluation

The three algorithms described above have different capabilities in terms of the concepts that they can represent. The first set of results in this section were generated by way of a synthetic dataset, and are used to highlight these differences. The second set of results were obtained from real data measured over the course of about ten months of operation of the Purdue University Network-Computing Hubs (PUNCH), during which time about five hundred PUNCH users executed approximately fifty thousand runs of various tools on shared compute servers connected to the Purdue Data Network. This set of results highlights the

³The distance metric does not have to satisfy the requirements for formal distance metrics [2].



Figure 1. Resource usage and prediction characteristics for a hypothetical tool with a single feature. The first plot shows the behavior of the tool with respect to the input parameter; the marked points indicate the instances stored in the knowledge base.

performance of the learning algorithms for real applications in a live, networked computing environment.

For illustrative purposes, consider a hypothetical tool whose resource-usage characteristics depend on a single feature. The first plot in Figure 1 shows the relationship between the input parameter (feature) and the (simulated) CPU time for this tool. The specific points marked in the plot represent the instances available in the knowledge base for each of the algorithms. The remaining plots show the characteristics of the predicted CPU time for the three learning algorithms. Observe that the nearest-neighbor and weighted-average algorithms are not able to learn the concept as well as the LLWR algorithm. The plots in Figure 2 show the effects of using a knowledge base with points that are relatively sparse and unevenly distributed — the results illustrate the higher sensitivity of the nearest-neighbor and weighted-average algorithms to the distribution of the observed instances.

In general, the nearest-neighbor and weightedaverage algorithms cannot track (even linear) polynomial surfaces without error [4]. This is illustrated in Figure 3, which shows the prediction errors for the onenearest neighbor (1-NN), three-point weighted-average (3-Avg), and locally linear LWR (LLWR) algorithms on a synthetic dataset. The dataset was made up of 1,000 instances with randomly-generated feature vectors. In addition to being able to reproduce linear sur-



Figure 2. Resource usage and prediction characteristics for a hypothetical tool with one feature and a knowledge base that is relatively sparse and has unevenly distributed data points. The marked points indicate the instances stored in the knowledge base.

faces without error, locally weighted regression algorithms can reproduce peaks and are insensitive to unsymmetrically distributed data [9, 24], making them an ideal choice for the domain.

In a computing environment, the performance of the learning algorithms can be evaluated in terms of two criteria: prediction error and prediction time. With respect to these criteria, two issues need to be addressed in order to make IBL algorithms suitable for extended use in a networked computing environment.

Basic IBL algorithms cannot track temporal variations in the concept to be learned, a feature that is crucial in a computing environment because systems can exhibit short-term fluctuations in performance. The solution to this problem is based on the observation that, if a run with a given feature vector is invoked at some time t, it (or a run with similar feature values) is likely to be invoked again at some time $t + \Delta t$. This temporal (and spatial) locality of runs is especially true in an academic environment, where a relatively large number of students tend to work concurrently on any given assignment. The PUNCH performance-modeling system employs a two-level knowledge base that allows the learning algorithms to exploit this locality. The first level of the knowledge base is used as a fixed-size cache, representing the short-term memory of the system. The second level acts as the long-term memory. Recently-observed instances are kept in the cache, and



Figure 3. Cumulative prediction error for IBL algorithms on a synthetic dataset with ten features. The nearest-neighbor and weightedaverage algorithms cannot track (even linear) polynomial surfaces without error.

are used preferentially in the process of making a prediction (see [18, 19] for details).

Minimizing the time required to predict resource usage is important because the predictions are made in real-time. Ideally, this time should be significantly smaller than (10% of, say) the shortest runs invoked by users. A bounded (and small) prediction time can be obtained by imposing an upper bound on the size of the knowledge base, in conjunction with the use of efficient search techniques. In PUNCH, the size of the toolspecific knowledge bases are constrained by selectively incorporating only *incorrectly predicted* feature vectors, and by discarding knowledge associated with feature vectors that have been consistently used to make incorrect predictions. This process is known as *instance* editing; additional details are available in [19].

The local learning algorithms described in this paper were tested on three semiconductor simulation tools (T-Suprem3, Minimos, and S-Demon). The datasets were constructed from trace data obtained by monitoring runs initiated by PUNCH on shared compute servers. This paper presents detailed results for T-Suprem3, a commercial package that simulates the processing steps used to manufacture silicon devices; results for the other datasets showed similar trends. The discussion focuses on the errors associated with the prediction of CPU time because of its importance in terms of scheduling.⁴





Figure 4. Effects of instance editing and caching on the learning system. Note the increase in the number of "zero-error" predictions with caching (c=5). Also observe the drop in lookup times and the size of the knowledge base with instance editing (iedit).

The feature vector for T-Suprem3 was made up of the following: 1) number of grid points, 2) total diffusion time, 3) cumulative epitaxial growth, 4) minimum implant energy, 5) number of deposit steps, 6) number of etch steps, and 7) number of implant steps.⁵ The learning instances collected for T-Suprem3 comprised of 8,100 runs whose CPU times ranged from 1 to 730 seconds (99.98% of the runs took less than 30 seconds). In the subsequent discussion, results obtained with and without instance editing are labeled **iedit** and **noedit**, respectively.

The top plot on the left hand side of Figure 4 shows the cumulative prediction errors of each of the local learning methods with instance editing and a cache of size five. The plot shows that the 1-NN algorithm learns faster than the other two algorithms. The middle plot shows the time required to retrieve instances from the knowledge base for each prediction of the 1-NN algorithm. This time is directly tied to the size of the knowledge base. As expected, the unmodified algorithm (noedit, c=0) results in monotonically increasing lookup time. Observe that the lookup time drops by almost a factor of two when a two-level knowledge base with a cache size of 5 is used (noedit, c=5). This is a clear indication of the temporal locality of runs and the corresponding usefulness of short-term memory. Applying instance editing results in a bounded knowledge base (iedit curves), indicating the effectiveness of discarding knowledge associated with predictable and noisy instances. The lookup time plots

time and network data-transfer time; memory and disk-space requirements will be predicted once the ongoing development of a monitoring system is complete.

⁵These features were identified by a domain expert.

for the three-point weighted-average and linear locally weighted regression algorithms were qualitatively identical. The bottom plot shows the distribution of the number of runs in terms of CPU time, along with a break up of the number of exact matches (i.e., when an identical feature-combination was found in knowledge base) and interpolated predictions for the locally weighted regression algorithm. Again, the increase in the number of exact matches with a cache validates the supposition of temporal locality. The corresponding results for the nearest-neighbor and weighted-average algorithms exhibit identical characteristics.

The table on the right hand side of Figure 4 presents a more detailed view of the effects of instance editing and caching. Consider the results for the 1-NN algorithm. The three rows correspond to the following conditions: noedit, c=0, iedit, c=0, and iedit, c=5. The first column shows the number of zero-error predictions. This quantity drops with instance editing because instances that would have helped for future queries are being discarded. Observe that the value increases again with caching. Indeed, a combined application of instance editing and caching results in a final value that is higher than the original because instance editing helps filter out the noise in the dataset while caching helps offset the disadvantages of discarding data. The average error is shown in the next column, illustrating that the error decreases with instance editing and caching. Finally, the last two columns show the effects of instance editing and caching on knowledge base size and lookup time. Instance editing drastically reduces the size of the knowledge base. Note that the majority of the runs ran in fewer than 5 seconds (see bottom plot in the figure), which probably contributes to the reduction of the knowledge base size. It is also interesting to note that the size of the resulting knowledge base is similar for all three algorithms, in spite of their different representational capabilities. When a cache of size five is added (third row for each algorithm), the knowledge base size does not grow by five. This implies that the most-frequently occurring instances and the best predictors of CPU time overlap to some degree. The behavior of the lookup time is explained by the fact that it is directly tied to the size of the knowledge base. Caching helps further reduce the lookup time even when the knowledge base is very small because the system first searches the cache.

The results for the other two algorithms show similar trends. The benefits of caching are independent of the specific algorithm because it exploits user-behavior, rather than the specific characteristics of individual learning algorithms.

7. Conclusions

Three instance-based learning algorithms nearest-neighbor, weighted-average, and locally weighted polynomial regression — were described in this paper. The algorithms are used to predict run-specific resource-usage on the basis of run-time input parameters supplied to the tool — to our knowledge, the performance-modeling system for PUNCH is the first to utilize automatically-extracted, tool-specific inputs in order to learn the resource-usage characteristics of tools.

A two-level knowledge base allows the learning algorithms to track short-term fluctuations in the performance of computing systems, and the use of instance editing techniques improves the scalability of the performance-modeling system. These two solutions significantly contribute to the suitability of IBL algorithms for extended use in a computing environment such as the one presented by a computational grid.

The results in this paper indicate that the nearestneighbor algorithm outperforms the more sophisticated locally weighted regression algorithm for the tools tested. However, this could be an artifact of the limitations of the current PUNCH environment: measured resource-usage often included a significant amount of noise (a better monitoring system is being implemented), and the distribution of run-times was more skewed towards shorter runs than what could be expected within a large computational grid environment. Further studies within larger computing environments with diverse tools and an extensive user-base are needed in order to draw a more general conclusion.

Acknowledgements

This work was partially funded by the National Science Foundation under grants MIPS-9500673, CDA-9617372, EEC-9700762, IIS-9733573, ECS-9809520, and EIA-9872516, and by an academic reinvestment grant from Purdue University.

References

- R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Proceedings of the 7th Heterogeneous Computing Workshop (HCW'98)*, Orlando, Florida, USA, 1998.
- [2] C. G. Atkeson, S. A. Schaal, and A. W. Moore. Locally weighted learning. AI Review, 11:11-73, 1997.
- [3] W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. Journal of the American Statistical Association, 74(368):829-836, 1979.

- [4] W. S. Cleveland, S. J. Devlin, and E. Grosse. Regression by local fitting: Methods, properties, and computational algorithms. *Journal of Econometrics*, 37:87-114, 1988.
- [5] W. S. Cleveland and C. Loader. Rejoinder to discussion of "smoothing by local regression: Principles and methods". In Statistical Theory and Computational Aspects of Smoothing, pages 113-120. Springer, 1996.
- [6] K. Deng and A. W. Moore. Multiresolution instancebased learning. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Montreal, Canada, 1995.
- [7] M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement based study on unix. *IEEE Transactions on Software Engineering*, 15(12):1579–1586, 1989.
- [8] A. B. Downey. Predicting queue times on spacesharing parallel computers. In Proceedings of the 11th International Parallel Processing Symposium (IPPS'97), Geneva, Switzerland, April 1997.
- [9] J. Fan and I. Gijbels. Local Polynomial Modelling and its Applications. Chapman and Hall, 1996.
- [10] I. Foster and C. Kesselman, editors. The GRID Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.
- [11] R. Freund, T. Kidd, D. Hensgen, and L. Moore. SmartNet: A scheduling framework for heterogeneous computing. In Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN-96), pages 514-521, 1996.
- [12] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel. Scheduling resources in multi-user, heterogeneous computing environments with SmartNet. In *Proceedings of the 7th Heterogeneous Computing Workshop (HCW'98)*, pages 184– 199, 1998.
- [13] K. Funahashi. On the approximate realization of continuous mappings by neural networks. Neural Networks, 2:183-192, 1989.
- [14] J. Gehring and A. Reinefeld. MARS a framework for minimizing the job execution time in a metacomputing environment. *Future Generation Computer Systems*, 12(1):87-99, 1996.
- [15] K. K. Goswami, M. Devarakonda, and R. K. Iyer. Prediction-based dynamic load-sharing heuristics. *IEEE Transactions on Parallel and Distributed* Systems, 4(6):638-648, 1993.
- [16] N. Kapadia, J. A. B. Fortes, and M. Lundstrom. The Computational Electronics Hub: A networkbased simulation laboratory. In Summary Record of the Workshop on Materials and Process Research and the Information Highway, page 31. National Academy Press, April 1996.
- [17] N. H. Kapadia. On the Design of a Demand-Based Network-Computing System: The Purdue University Network-Computing Hubs. PhD thesis, Department of

Electrical and Computer Engineering, Purdue University, August 1999.

- [18] N. H. Kapadia, C. E. Brodley, J. A. B. Fortes, and M. S. Lundstrom. Resource-usage prediction for demand-based network-computing. In *Proceedings of the 1998 Workshop on Advances in Parallel and Distributed Systems (APADS)*, West Lafayette, Indiana, October 1998.
- [19] N. H. Kapadia, C. E. Brodley, J. A. B. Fortes, and M. S. Lundstrom. Resource usage prediction for demand-based network-computing. Technical Report TR-ECE 98-9, Department of Electrical and Computer Engineering, Purdue University, 1998.
- [20] N. H. Kapadia and J. A. B. Fortes. On the design of a demand-based network-computing system: The Purdue University Network-Computing Hubs. In Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC'98), pages 71-80, Chicago, Illinois, July 1998.
- [21] N. H. Kapadia and J. A. B. Fortes. The PUNCH network desktop. Technical Report TR-ECE 99-1, Department of Electrical and Computer Engineering, Purdue University, 1999.
- [22] A. W. Moore, J. Schneider, and K. Deng. Efficient locally weighted polynomial regression predictions. In Proceedings of the 1997 International Machine Learning Conference, Nsahville, Tennessee, 1997.
- [23] W. W. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, 2nd edition, 1992.
- [24] S. Schaal. Nonparametric regression for learning. In Proceedings of the Conference on Adaptive Behavior and Learning, pages 123-133, Center for Interdisciplinary Research, University of Bielefeld, Germany, 1994.
- [25] S. Schaal and C. G. Atkeson. Assessing the quality of learned local models. In J. Cowan, G. Tesauro, and A. J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 160–167. Morgan Kaufmann, 1994.
- [26] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In Proceedings of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.
- [27] A. Svensson. History, an intelligent load sharing filter. In Proceedings of the 10th International Conference on Distributed Computing Systems, pages 546-552, 1990.
- [28] C.-J. Wang, P. Krueger, and M. T. Liu. Intelligent job selection for distributed scheduling. In Proceedings of the 13th IEEE International Conference on Distributed Computing Systems, pages 517-524, 1993.