Resource-Usage Prediction for Demand-Based Network-Computing

Nirav H. Kapadia, Carla E. Brodley, José A. B. Fortes, and Mark S. Lundstrom
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907-1285
{kapadia, brodley, fortes, lundstro}@ecn.purdue.edu

Abstract

This paper reports on an application of artificial intelligence to achieve demand-based scheduling within the context of a network-computing infrastructure. The described AI system uses tool-specific, run-time input to predict the resource-usage characteristics of runs. Instance-based learning with locally weighted polynomial regression is employed because of the need to simultaneously learn multiple polynomial concepts and the fact that knowledge is acquired incrementally in this domain. An innovative use of a two-level knowledge base allows the system to account for short-term variations in compute-server and network performance and exploit temporal and spatial locality of runs. Instance editing allows the approach to be tolerant to noise and computationally feasible for extended use. The learning system was tested on three tools during normal use of the Purdue University Network Computing Hubs. Results indicate that the described instance-based learning technique using locally weighted regression with a locally linear model works well for this domain.

1. Introduction

A demand-based computing system can be characterized by its universal accessibility and its ability to make automatic cost/performance tradeoff decisions at run-time. Universal accessibility can be provided via a widely-used networked interface such as the world-wide web. Run-time cost/performance tradeoff decisions require that the infrastructure be able to decide how (which implementation - e.g., sequential versus parallel) and where (which platform) to run a tool. This paper presents an application of artificial intelligence to achieve *demand-based scheduling* within the context of a network-computing infrastructure (the Purdue University Network Computing Hubs, or PUNCH [6, 7]) that allows users to access and run existing software tools via standard world-wide web browsers.

Cost/performance tradeoff decision are based on run-specific resource requirements and tool-specific portability information. While portability information is usually available a priori, run-specific resource requirements generally depend on the run-time input to the tool. Although it may sometimes be possible to obtain analytical expressions that describe the relationship between the run-time input and the corresponding resource usage (e.g., matrix-manipulation codes), in general, tools tend to exhibit complex behavior that make such analytical expressions nearly impossible. Even when it is possible to determine an analytical expression, the resource-usage characteristics cannot be computed from an expression that simply describes the computational complexity of the algorithm; the appropriate architecture-specific constants must also be determined.

To our knowledge, this is the first system to use tool-specific, run-time inputs to predict the resource usage characteristics of runs. Other work on resource-usage prediction (e.g., [3, 4, 9, 10, 11]) utilizes tool-specific analytical expressions or statistical data obtained from past runs (e.g., average execution time) to predict future resource usage. Results show that such heuristics can be used to identify better schedules. These approaches were not used here because the resource-usage characteristics of the tools in our domain are highly dependent on run-specific parameters (e.g., a Monte Carlo simulation may execute for anywhere from a few minutes to several days, depending on the inputs).

The AI system described here uses instance-based learning to predict the CPU time and the network data-transfer time for a given run on the basis of the associated run-time input.[1] An innovative use of a two-level knowledge base allows the system to account for

[1] Memory and disk-space requirements are not predicted at the moment. These parameters will be predicted once the ongoing development of a monitoring system is complete.

short-term variations in compute-server and network performance and exploit temporal and spatial locality of runs. Instance editing allows the approach to be tolerant to noise and computationally feasible for extended use. The learning system was tested on three semiconductor simulation tools during normal use of PUNCH in Fall 1997, and on four synthetic datasets (off-line). Locally weighted polynomial regression with a locally linear model was found to perform well for all the datasets tested.

2. PUNCH

The Purdue University Network Computing Hubs (PUNCH)[2] is a distributed network-computer that provides geographically dispersed users with universal, web-based access to tools [6, 7]. Functionally, it allows users to: a) upload and manipulate input-files, b) run programs, and c) view and download output - all via standard WWW browsers. Currently, PUNCH consists of four discipline-specific hubs that contain tools from semiconductor technology, VLSI design, computer architecture, and parallel processing. A fifth hub is devoted to tools that were developed with support from the Semiconductor Research Corporation (SRC). These hubs contain over thirty tools from eight universities and four vendors and serve more than 500 users from within Purdue, across the US, and in Europe.

PUNCH allows on-demand management of existing software and hardware resources by delaying the binding of a user's command to a specific implementation and machine until run-time, at which point the requirements of the given run can be analyzed. The resource-requirements of a particular run are determined by PUNCH's AI sub-system, which qualifies the user-supplied tool-input with available tool-specific scalability and portability information. The output of the AI system is used to match a user's request with the underlying network-accessible tools and resources.

3. Domain Characterization

3.1. Tool Characteristics

The tools available on PUNCH come from a wide variety of environments and disciplines. Each tool requires its own set of features and a separate knowledge base. In general, establishing the correct (i.e., relevant) features for a given tool is a difficult problem - realistic tools tend to use sophisticated algorithms whose behavior cannot be easily correlated to the user-supplied input values. Another problem associated with the feature vector is that the range of values that a given feature can assume is generally not known a priori, particularly in a research environment. In terms of the artificial intelligence system, these issues require that the system be able to: a) ignore irrelevant features, b) detect inadequate feature vectors, and c) work with unscaled features.[3]

The relationship between the n inputs supplied to a program and the corresponding resource-usage characteristics is defined by a set of polynomials in n-dimensional space. Thus, the learning algorithm used for this domain must be able to capture concepts described by (possibly multiple) polynomial functions. Moreover, this relationship often has a non-deterministic component with respect to the available inputs. For example, the convergence rate of an iterative matrix-manipulation algorithm is likely to depend on the distribution of the eigenvalues of that matrix, which are difficult to compute in advance. This effectively implies that the learning algorithm will have to work with an incomplete or noisy description of the features that determine the resource-usage characteristics of the program.

3.2. Run-Time Environment

When a request for a run is received by PUNCH, it extracts the values of the administrator-specified features from the user-supplied input and uses them to predict the resource-usage characteristics. The prediction is then used to determine how and where to schedule the request. After the run completes, PUNCH provides the true resource-usage characteristics to the artificial intelligence system, allowing the learning algorithm to incorporate the new information into its knowledge base. Because this process happens in real-time and during normal use of the system, an incremental learning approach is needed. The run-time environment is also interactive, which requires the predictions to be made in real-time. This in turn implies that the resources used by the artificial intelligence system cannot grow monotonically with time.

The final issue that affects learning is short-term variations in the performance of computer systems. Short-term variations in performance can occur due to unpredictable events such as a file-server or network router becoming overloaded. The learning algorithm must be able to quickly tailor its predictions to such short-term variations without being unduly affected by them in the longer term.

[2] The Purdue University Network Computing Hubs can be accessed at "http://www.ecn.purdue.edu/labs/punch/". Courtesy accounts are available.

[3] "Unscaled" in this context implies that the system cannot use a constant scaling factor that has been determined a priori; it can still scale features on the fly.

4. The Artificial Intelligence System

4.1. Algorithm Selection

Instance-based learning (IBL) algorithms approximate the target concept by dividing the input space into many partitions, each of which is approximated by an independent function $y_i = f_i(x, \theta_i)$ [8]. IBL algorithms do not require an explicit training phase, and, because of their localized nature, they are relatively insensitive to the structural complexity of the function to be learned [8].

There are many instance-based learning algorithms, including nearest neighbor, weighted average (kernel regression), and locally weighted regression techniques. Locally weighted regression (LWR) fits a surface to nearby points, typically via a locally linear (LLWR) or quadratic (QLWR) model.[4] With a linear (quadratic) model, the target concept is locally approximated by a linear (quadratic) surface. Recall that global regression [12] constructs a surface that minimizes the sum of the squares of the errors. In contrast, locally weighted regression [2] 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 weighting function is chosen so as to eliminate the effects of remote datapoints. The size of the local neighborhood (i.e., the region in which the weights are non-zero) is called the *kernel width* or *bandwidth*.

In addition to being able to reproduce linear surfaces without error, locally weighted regression algorithms can reproduce peaks and are insensitive to unsymmetrically distributed data [1, 8]. This makes locally weighted regression an ideal choice for the given domain. The locally linear model is chosen over the locally quadratic model for two reasons: a) it learns faster (for a locally linear surface), and b) it requires less time to make a prediction [5].

4.2. Learning Issues

The basic LLWR learning algorithm addresses the following issues: a) learning sets of polynomial functions, b) incremental learning, and c) support for irrelevant and unscaled features. Modifications are required to address: a) detection of inadequate feature vectors, b) short-term variations, c) noisy features, and d) scalability of the knowledge base during extended use. The scalability issue is the most critical because the basic IBL algorithms do not scale well enough for extended use in the PUNCH environment (the size of the knowledge base and the average per-prediction lookup time increase monotonically with the number of runs).

[4] Higher-order local models are generally not used because of the associated computational cost.

The subsequent sections present solutions for each of the mentioned problems. Detection of inadequate feature vectors is addressed by storing appropriate meta-information about the instances in the knowledge base [5]. Sensitivity to short-term variations without an associated loss in longer-term performance is obtained by using a two-level knowledge base, which also helps the IBL algorithms scale better. Finally, scalability and noise issues are addressed by: a) not adding all instances to the knowledge base, and b) allowing instances to be discarded from the knowledge base.

4.3. Knowledge-Base Organization

A *two-level knowledge-base* organization is selected on the basis of the two considerations outlined below; the first level of the knowledge base acts as a fixed-size cache, representing the *short-term memory* of the system. The considerations are: a) short-term variations in the behavior of computing resources, and b) temporal and spatial locality of runs. For this domain, the principle of temporal locality can be stated as follows. If a run with a given feature vector is invoked at some time t, it is likely to be invoked again at some time $t + \Delta t$. This is especially true in an academic environment, where a relatively large number of students tend to work concurrently on any given assignment. Similarly, the principle of spatial locality can be stated as follows: if a run with a given feature vector is invoked at some time t, runs with similar feature vectors are likely to be invoked in the near future. This assumption applies to users who, for example, need to characterize a system by perturbing a few parameters at a time (characteristic of a research environment).

4.4. Knowledge Retrieval

In order to retrieve the resource-usage characteristics of a given feature vector, the learning algorithm scans the two-level knowledge base in the following manner. It first looks in the cache for an exact match to the query in terms of the feature vector. If a match is found, the algorithm makes its prediction on the basis of the precomputed characteristics associated with that feature vector. If a match is not found, the process is repeated with the second level of the knowledge base. The time-associated performance advantages of the two-level organization are based on the supposition that a match will be found in the cache for a significant number of requests.

If an exact match is not found in either level of the knowledge base, the learning algorithm retrieves the $2 \times (n+1)$ feature vectors (n is the length of the feature vector) that are closest to the query and uses them for

the locally weighted regression analysis. Recall that a linear polynomial with n unknowns contains $n + 1$ terms. This implies that, at a minimum, $n + 1$ feature vectors are required to obtain a unique solution to the query (with linear LWR). Using twice as many instances provides a degree of tolerance for noisy and "linearly dependent" (e.g., datapoints that have identical values in a given dimension do not provide any information about that dimension) datapoints. The closeness of feature vectors is evaluated in terms of Euclidean distances in a co-ordinate system that is normalized with respect to the query point. The normalization is necessary because of the unpredictable nature of the range and distribution of the feature values.

4.5. Knowledge Management Policies

Basic local learning techniques incorporate all instances into the knowledge base. This results in a monotonically increasing knowledge base, making the AI system unscalable. There are two ways to address this problem. The first option is to selectively incorporate only incorrectly predicted feature vectors into the knowledge base. The second option is to discard knowledge associated with feature vectors that have been consistently used to make incorrect predictions.

The first option can be expected to work well in situations where the learning algorithm is able to capture the target concept. Note that the concept only has to be *locally* linear for LLWR to capture it; its global structure can be much more complex. Selectively incorporating knowledge could have a negative impact on the learning rate of the system. For example, in some cases, an instance that was discarded in the past could have resulted in a better prediction for the current query. Although this problem cannot be completely eliminated, it is partially addressed by the two-level knowledge base. All instances are incorporated into the cache regardless of the current policy. When the cache overflows, instances are incorporated into the second level according to the current policy.

With the second option, knowledge associated with feature vectors that have consistently (more than 50% of the time, say) been used to make incorrect (off by more than 10%) predictions is discarded. The keep/discard decisions are made periodically; feature vectors that do not have adequate use statistics associated with them are allowed to retain their history for the next time-frame (the history is reset once a keep/discard decision is made). In practice, the described policy is not enforced until after a certain number of runs have been observed, to allow for errors before the concept is learned.

Finally, to account for situations in which these two heuristics fail, the size of the knowledge base has a hard upper bound associated with it. When the size exceeds a specified threshold, a LRU policy is used to discard feature vectors. In the experiments described in this paper, the upper bound (100) was never reached.

5. Experimental Evaluation

In this application, there are three performance criteria: a) the prediction error, b) the time required for prediction, and c) the growth-rate of the knowledge base. The learning system was tested on three semiconductor simulation tools (T-Suprem3, Minimos, and S-Demon) during normal use of PUNCH in Fall 1997. Runs consisted of simulations for class projects and homework assignments. The learning instances collected for T-Suprem3, Minimos, and S-Demon comprised of 3398, 966, and 131 runs, respectively. The system was also tested with four synthetic datasets.

This paper presents detailed results for T-Suprem3 (a commercial device-fabrication simulator from Technology Modeling Associates, Inc.); results for the other datasets showed similar trends. The features associated with T-Suprem3 characterize aspects of the fabrication process of a semiconductor device. Specifically, the feature vector was made up of the following: a) number of grid points, b) total diffusion time, c) cumulative epitaxial growth (in terms of thickness), d) minimum implant energy, e) number of deposit steps, f) number of etch steps, and g) number of implant steps.

5.1. Results

The results in this section focus on the errors associated with the prediction of CPU time because of its importance in terms of scheduling. For convenience, the different policies described earlier are named as follows. The basic IBL approach using LLWR is called `basic`. The policy that does not add accurately predicted feature vectors into the knowledge base is called `noadd`. The policy that deletes feature vectors that consistently result in bad predictions from the knowledge base is called `noisetol`. Finally, the combined application of `noadd` and `noisetol` is called `combined`.

The cumulative prediction error plots associated with the `basic` policy (Figure 1) confirm that the AI system is able to learn the relationship between the run-time inputs and the resource-usage characteristics of T-Suprem3. Error prediction plots for the other policies show similar trends; the final cumulative errors for all policies are shown in Table 1. Observe that discarding feature vectors that consistently result in incorrect predictions (`noisetol` and `combined` policies) considerably improves the prediction accuracy. Also note

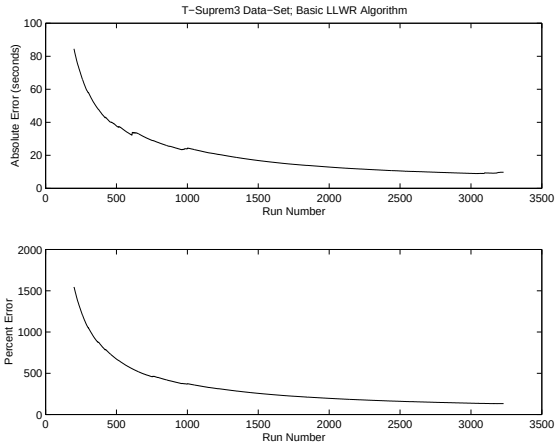

Figure 1. Prediction error - `basic` policy.

Table 1. Cumulative prediction error.

Policy	Cache Size = 0		Cache Size = 20	
	Abs	Percent	Abs	Percent
`basic`	9.77	131.97	9.77	131.97
`noadd`	9.36	126.93	9.06	119.68
`noisetol`	5.78	46.58	5.72	45.01
`combined`	6.37	62.46	6.26	50.77

that the prediction error associated with the `noadd` policy is essentially the same as that of the `basic` policy, indicating that discarding feature vectors did not have a negative impact on the system's overall ability to learn. Finally, the data in the table shows that caching reduces the prediction error for the `noadd` and `combined` policies, which do not add (to the knowledge base) feature vectors whose resource-usage characteristics can be predicted accurately. More detailed analysis of the results shows that this is especially true for short runs [5].

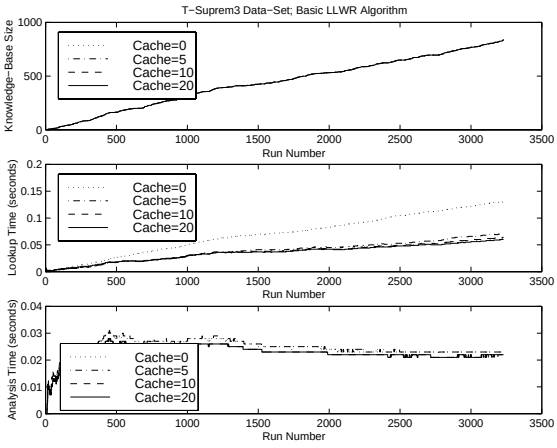

Figure 2. System scalability - `basic` policy.

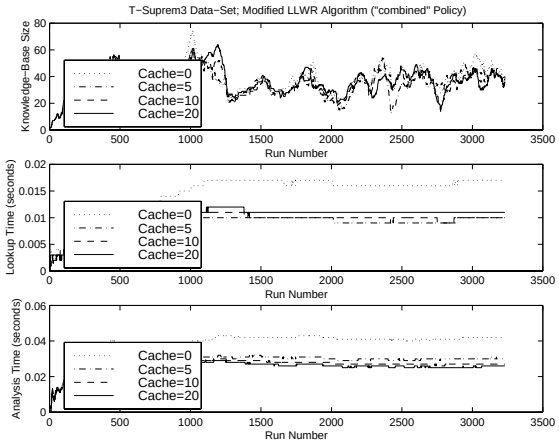

Figure 3. System scalability - `combined` policy.

The top plots in Figures 2 and 3 show the growth rate of the knowledge base for the `basic` and `combined` policies, respectively. As expected, the knowledge base grows monotonically for the first policy and is bounded for the second one. The oscillations in the size of the knowledge base (`combined` policy) are caused by the periodic deletion of noisy feature vectors from the knowledge base. Also observe that the size of the knowledge base is dramatically smaller than that for the `basic` policy and that the short-term memory does not significantly affect the overall size of the knowledge base. The lookup time shown in the middle plots is a function of the size of the knowledge base, and consequently tracks the top plots. The plots show the beneficial effects of the two-level knowledge base in terms of reduced lookup times. The analysis time shown in the bottom plots is equal to the time required to compute the regression matrices if a matching feature vector is not found in the knowledge base. If a match is found, the analysis time is zero. The average analysis time is only dependent on the number of feature vectors used for regression, which is a bounded value. Consequently, once an adequate number of feature vectors are available, the analysis time is approximately constant. For the `combined` policy, caching also helps reduce the analysis time. This is a consequence of a smaller number of interpolated queries, which, in turn, is an indication of the temporal locality of runs. (A reduction in the number of interpolated queries is accompanied by a corresponding increase in the number of exact matches.)

6. Conclusions

Our results indicate that the described instance-based learning approach using locally weighted regres-

sion works well for the domain considered in this paper. Selectively adding feature vectors into the knowledge base and discarding feature vectors that consistently result in inaccurate predictions make the described learning approach scalable and tolerant to noise. Experimental data collected during normal use of PUNCH validates the assumptions of temporal and spatial locality. The use of a two-level knowledge base, which exploits these assumptions, results in reduced prediction error, faster retrieval of feature vectors, and smaller (average) analysis time.

The ideas behind the multi-level knowledge base and instance editing are not limited to locally weighted regression; they can be applied to any local learning algorithm to address scalability and noise tolerance issues. Ongoing work is directed at evaluating the benefits of caching and instance editing for nearest-neighbor and weighted-average learning algorithms.

7. Future Work

The logical extension to the described work is to apply it to a larger number of tools. The described prediction techniques do not make any domain-specific assumptions with respect to tools. Consequently, it should be possible to apply them to any tool for which there is a correlation between the input parameters and the corresponding resource usage.

The measured network data-transfer time currently consists of the time required to move data to and from the execution platform and the associated file servers. The completion of the ongoing development of a monitoring system will allow the CPU and message-passing times for individual tasks of parallel programs to be measured and predicted.

The current implementation of the AI system does not account for the heterogeneity of hardware resources. In order to do this, the system will have to learn (tool-specific) scaling factors for each architecture, in addition to the resource usage characteristics.

Yet another goal is to exploit the predictability of long-term resource-usage trends. Demands on computational resources tend to follow patterns that can be learned (certain resources tend to be heavily loaded during the late afternoon hours, for example). An AI-based approach to resource allocation could exploit this fact and learn an *anticipatory* scheduling policy.

Acknowledgements

This work was partially funded by the National Science Foundation under grants CDA-9617372, EEC-9700762, and MIPS-9500673. The feature vectors for T-Suprem3 and Minimos were identified with the help of Mike Young and Steven Bourland, respectively.

References

[1] W. S. Cleveland and S. J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[2] J. Fan and I. Gijbels. *Local Polynomial Modelling and its Applications*. Chapman and Hall, 1996.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] N. H. Kapadia, J. A. B. Fortes, and M. S. Lundstrom. The Semiconductor Simulation Hub: A network-based microelectronics simulation laboratory. In *Proceedings of the 12th Biennial University Government Industry Microelectronics Symposium*, pages 72–77, July 1997.

[8] 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.

[9] J. M. Schopf and F. Berman. Performance prediction in production environments. In *Proceedings of the First Merged Symposium: 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*, Orlando, Florida, March 1998.

[10] 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.

[11] A. Svensson. History, an intelligent load sharing filter. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 546–552, 1990.

[12] R. J. Wonnacott and T. H. Wonnacott. *Regression: A Second Course in Statistics*. John Wiley and Sons, 1981.