Accepted for submission by CCC '97

Structural Prediction Models for High-Performance Distributed Applications

Jennifer Schopf
Department of Computer Science and Engineering
University of California San Diego
Email: jenny@cs.ucsd.edu

Abstract

We present a structural performance model that uses application profiles and component models to predict an application's performance on a set of distributed resources. We decompose application performance in accordance with the structure of the application: that is, into interacting component models that correspond to component tasks. Then, using the application profile and available information as guides, we select models for each component appropriately. As a proof of concept, we have implemented this approach for two distributed applications, a master-slave genetic algorithm code and a red-black stencil successive over-relaxation code. Our predictions are within 10% of actual time.

Context

Clusters of distributed machines have become a common platform for high performance applications, but remain a challenging environment in which to achieve good performance. One reason for this is the difficulty of predicting an application's execution time in this variable setting, where only minimal information may be available. In particular, as a part of the AppLeS scheduling project [BW96, BWF+96], we are interested in estimating the performance of applications for use in dynamic scheduling. Towards this end, we build structural models, where a model can be thought of as a black box that takes as inputs a problem size and a resource set and predicts execution time. Because this environment combines varying non-dedicated resources and possibly dissimilar implementation paradigms, many established modeling techniques do not satisfy the needs of our scheduler.

Our approach builds models that mirror the structure of the application by breaking the overall prediction model into coarse-grain components; individual models are then chosen to fit the characteristics of these components. This allows for the variability of this environment, where different application components may need different models of performance. This approach also provides a method by which different structural components can be modeled to differing levels of detail as needed.

From conversations with application developers working with distributed parallel applications [ASW+95], we know that users can easily supply information about code structure and profile. We have tried to allow for a variety of different sources of application data of varying qualities and use. As in any setting, more (and better) data will lead to better predictions, so our approach also allows the upgrading of information if initial data leads to flawed predictions. Examples of the possible data to be supplied would be a time per element estimate for the computation components (in our experience, these are achieved most often through operation counts or benchmarks), message sizes and bandwidth or latency information for communication components, and size of data and memory speeds for memory components. This level of information is generally available to some degree of accuracy since distributed parallel codes are often highly tuned.

Our Approach

Informally, we follow four steps to build our prediction models. First, we examine the structure of the application and construct a top-level model to represent this structure. This gives us a high-level equation representing the functionality of the execution.

Second, we need an application profile that describes where execution time for the application is spent. Admittedly, this is a difficult value to define exactly, as it will depend on many factors: architecture, specific implementations of the application, load on relevant machines, problem size, etc. In practice, these values can be found through profiling [GKM82] or timing runs of the application, or an estimated value can be supplied by the application developer. Often, these estimates are accurate enough for our purposes.

Third, we use the application profile to guide the choice of component models. For example, if, as is often the case, a profile indicates that one component dominates the execution time, we can begin by modeling only this component. If the overall prediction is too erroneous, examine additional components as needed. The component models are defined, possibly recursively, as combinations of input parameters (benchmarks, constants, arithmetical operations, etc.) and/or other component models. Their selection primarily depends on available sources of information.

Lastly, we analyze the accuracy of the model and determine a course of action. The degree of accuracy needed by a prediction model varies by the use of the model. If our model is not sufficiently accurate for its use, there are several transformations we can perform:

- Change a part of a component model. This is done to avoid using input data that is inaccurate.
- Refine a component model by increasing the detail. This is done when an execution property is not being modeled within a component: for example, memory hierarchy behavior or communication contention over a shared network.
- Add other component models. This is done in accordance with the profile, or when there is evidence that the profile is in error.
- Re-structure the top-level model. This is done when tasks or task interactions not being captured by the current model approach.

Evaluating the exact cause of an error in a prediction involves making use of additional information about the input data. Several forms of meta-data are common: variance information showing how values may range in time, confidence factors demonstrating a developers confidence in the accuracy of the data, or history information if the values are input dynamically. In addition, if a given application has been modeled previously, there may be information about the component values used and their accuracy. This is the subject of our current research.

Developing a Performance Model for a Master-Slave GA

Genetic algorithms (GA) form a class of applications originally developed by the artificial intelligence community as an optimization technique for NP-complete and NP-hard problems. As such, they are now being used by several groups of computational scientists [DTM+96, Schulze96, PM94] to address problems such as protein folding. Our distributed implementation uses a global population with synchronization between generations [Bhatia96]. It was implemented in C using PVM on a heterogeneous cluster of workstations in the Parallel Computation Laboratory (PCL) at UCSD.

The goal of our experiments is to show the feasibility of this modeling approach. This is, given a minimal amount of information tuned to the structure of the application, could we predict the application's performance accurately?

Stepping through our approach, we first define the structure for the model. This application is a typical example of a distributed master-slave. The master process broadcasts (using a PVM multicast routine) a global population to the slaves, which compute the next generation of solutions independently of one another. The data from the slaves is then gathered by the master. The master performs some analysis on the collected data to determine the next generation, and this process is repeated. So we have a top-level structural model:

$$\mathbf{ExTime} = \mathbf{MasterComp} + \mathbf{Scatter} + \mathbf{Max}[\mathbf{Slave_i Comp}] + \mathbf{Gather} \tag{1}$$

For step two, the application developer gave us an application profile that showed the majority of the execution time was spent in the SlaveComp portion of the application.

In step three, we refine component models. Following the application profile, we decided to concentrate our modeling effort on the SlaveComp section of the model and analyzed two fairly detailed choices for this component. Our first choice, driven by the availability of information, was to model the slave computation using an operation count model.

Step four involves evaluating the results. Our choice of an operation count model lead to inaccurate predictions due to the fact that the inner loops of the GA code are non-deterministic and irregularly structured; therefore it was difficult to derive an accurate operation count statically. So we chose another component model that avoided the faulty data, and, using benchmarks for a median problem size, achieved results within 10% of the actual run time. We evaluated several different models for the communication routines, but none contributed statistically significant amounts to the predictions as a whole. Communication costs were low, in part because the experiments were run in isolation so there was no extraneous network traffic. Communication costs would contribute more if latency were higher, or contention caused considerable communication delay. If this were the case, we would need to model both the scatter and the gather routines as well.

Developing a Performance Model for a Red-Black SOR

We achieved similar results using a red-black successive over-relaxation (SOR) algorithm that solves Laplace's equation. These are detailed in the full paper.

Conclusion

We have presented a new structural approach to modeling cluster applications, and two proof of concept examples. Both real applications require flexible models to predict their behavior, but the choice of models is not obvious. Our paradigm is well suited to this decision.

References

ASW+95 Anglano, C., Schopf, J., Wolski, R. and Berman, F., Zoom: A Hierarchical Representation for Heterogeneous Applications, UCSD CS Tech Report #CS95-451, 1995.

Bhatia96 Bhatia, Karan. Personal communication. 1996.

BW96 Berman, F., and Wolski, R., Scheduling from the Perspective of the Application. In Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, 1996.

BWF+96 Berman, F., Wolski, R., Figueira, S., Schopf, J., and Shao, G.. Application-Level Scheduling on Distributed Heterogeneous Networks. In Proceedings of Supercomputing '96, 1996.

DTM+96 Deaven, D., Tit, N., Morris, J., and Ho, K. Structural Optimization of Lennard-Jones Clusters by a Genetic Algorithm. Chemical Physical Letters 256, pp. 195, 1996.

GKM82 Graham, S., Kessler, P., McKusick, M., gprof: A Call Graph Execution Profiler, Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

PM94 Pedersen, Jan and Moult, John. Determination of the Structure of Small Protein Fragments using Torsion Space Monte Carlo and Genetic Algorithm Methods. In Proceedings of Meeting on Critical Assessment of Techniques for Protein Structure Prediction, Asilomar Conference Center, December 1994.

Schulze96 Schulze-Kremer, S. http://www.techfak.uni-bielefeld.de/bcd/Curric/ProtEn/ proten.html, 1996.