Active Harmony: Towards Automated Performance Tuning

Cristian Ţăpuş, I-Hsin Chung, and Jeffrey K. Hollingsworth

crt@cs.caltech.edu, ihchung@cs.umd.edu, hollings@cs.umd.edu

Department of Computer Science, and Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract

In this paper we present the Active Harmony automated runtime tuning system. We describe the interface used by programs to make applications tunable. We present the Library Specification Layer which helps program library developers expose multiple variations of the same API using different algorithms. The Library Specification Language helps to select the most appropriate program library to tune the overall performance. We also present the optimization algorithm that we used to adjust parameters in the application and the libraries. Finally, we present results that show how the system is able to tune several real applications. The automated tuning system is able to tune the application parameters to within a few percent of the best value after evaluating only 11 configurations out of over 1,700 possible combinations.

1. Introduction

Applications are no longer monolithic programs written for a specific purpose. Instead, most software today makes extensive use of libraries and re-usable components. This approach generally results in software that is faster to build and more modular. However, one problem with this approach is that the various libraries used by an application are not tuned to the specific application's need. In addition, applications are frequently used in very different ways. For example, different users may employ a single commercial simulation application for radically different types of simulations. As a result of this reuse of software, applications may not run well in all configurations.

The transient, rarely repeatable behavior of Grid [4] computing environment indicates the need to replace standard models of post-mortem performance optimization with a real-time model, one that optimizes application and runtime behavior during program execution. Automatic program library selection provides a framework to help with this goal; it helps to tune the application during runtime execution by monitoring the underlying library performance and switching underlying program library as needed. This is an important step toward automated performance tuning in the Grid computing.

To meet the needs of this type of computing environment, we have been developing the Active Harmony system that allows runtime switching of algorithms and tuning of library and application parameters. We have also developed a set of runtime tuning algorithms that help to intelligently set these parameters at runtime to tune the overall performance of an application.

Active Harmony is an infrastructure that allows applications to become tunable by applying very minimal changes to the application and library source code. This adaptability provides applications with a way to improve performance during a single execution based on the observed performance. The types of things that can be tuned at runtime range from parameters such as the size of a read-ahead parameter to what algorithm is being used (e.g., heap sort vs. quick-sort).

2. System Design

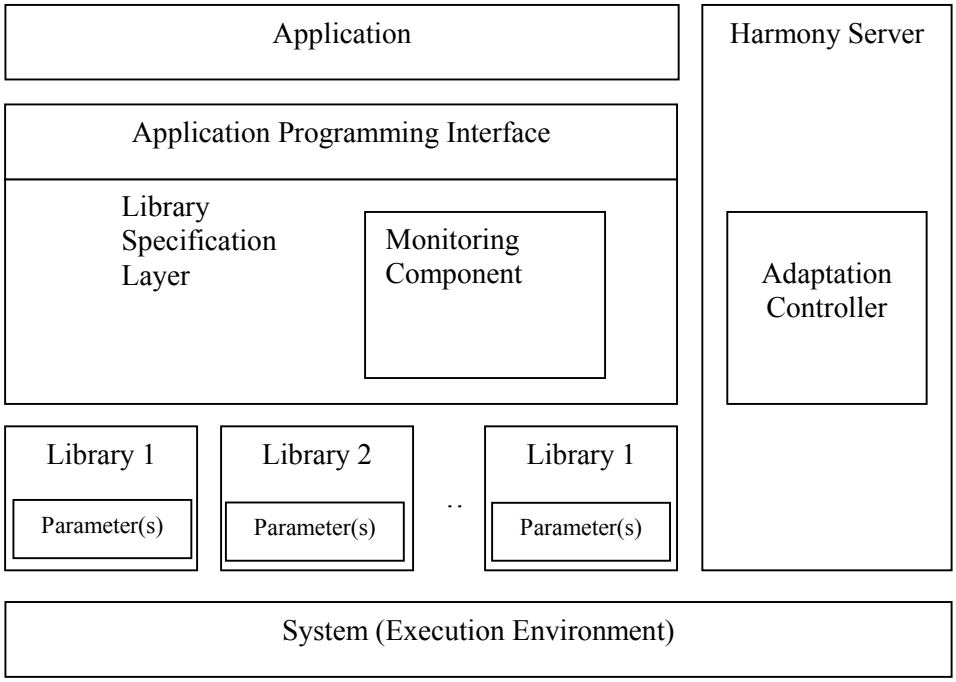

Figure 1: Active Harmony automated runtime tuning system

Figure 1 shows the Active Harmony automated runtime tuning system. The Library Specification Layer provides uniform API to library users by integrating different libraries with the same or similar functionality. This layer uses the Harmony Controller to select among different implementations of the library. The library specification layer also monitors the performance of the library to improve the decision for future usage of the program library.

The Adaptation Controller is the main part of the Harmony server. The Adaptability component manages the values of the different tunable parameters provided by the applications and changes them for better performance. The Adaptation Controller is written in Tcl for an easier interpretation of the information provided by applications and resources.

3. Library Specification Layer

The role of the library specification layer is to help the application use the most appropriate underlying algorithm. To achieve such a goal, it first characterizes the request from the application and monitors the performance of underlying program libraries. Based on the collected information, it will redirect the function calls to the selected underlying program library.

Selection among the available libraries is done via the Adaptation Controller. During the execution, the library specification layer will send the characteristics of requests to the Adaptation Controller. The Adaptation Controller returns the suggested underlying algorithm to use according to the result of its decision process. In the current implementation, when the Adaptation Controller is selecting an algorithm, it tries to explore all possible algorithms for at least a brief period of time. Based on observed performance, an appropriate algorithm is selected.

The performance metrics commonly used are the usage of resources by the program library such as CPU time or memory space. Library developers can specify multiple program library performance metrics in the library specification language. The underlying program libraries have to provide function calls in their API to support the measurement as well as the estimation of these performance metrics. Selecting appropriate underlying program library is the role of the Adaptation Controller. In the current implemen-

tation, the Adaptation Controller tries to minimize the value of the first performance metric when searching for appropriate underlying program library.

The key idea of algorithm selection is that usage patterns of a library may require different types of requests underlying algorithms or data structures. For example, in an implementation of a table abstraction, the data structure used and the workload pattern will both affect the performance. In a workload with high search rate and high data element location density, arrays would outperform linked lists. However, if the data element density is sparse and memory space is critical, the linked data structure should be chosen. The characteristics of requests play an important role in selecting appropriate underlying program library. Library developers specify possible characteristics of how use of an API could favor one implementation of the functionality over another. Those request characteristics can be either variables with primitive data types or expressed by basic Boolean operations on those variables.

The Library Specification Language currently supports libraries written in both C and Fortran. The Library Specification Language generates header files that interpose glue code to allow libraries (or algorithms) with slightly different calling conventions to be integrated into a uniform API for upper layer users. It also provides the indirection to allow runtime switching among the different implementations. The runtime switching code includes the ability for library writer to specify mapping functions that can change the underlying data structures (such as going from a dense to sparse matrix representation).

4. The Harmony Parameter API

In order to allow the Harmony server to change library or application parameters, we have developed a library of functions that register tunable parameters and provide ways for the code to get the new parameters from the Harmony Adaptation Controller. The changes required to make a program tunable using this interface is relatively small. For many programs we have "harmonized" the change amounted to less than 50 lines of code. The full paper includes an example of specifying application and library parameters using the Harmony API.

5. Parameter Tuning Algorithm

In an earlier version of the Harmony system [7], we had a simple greedy algorithm to handle automatic selection of the appropriate parameters. However, for larger applications a more sophisticated algorithm is needed.

The problem of selecting good parameters reduces to finding a k-tuple in the value space determined by the values of the tuning parameters specified by the application, such that the application performs best. If we consider that better performance is represented by a smaller value of the performance function the problem a minimization problem.

However, the problem is more complex due to the nature of the value space and that of the performance function. For example, a simple performance function could be the time spent by an application to complete a certain task. However, the value of this performance function depends not only on declared application parameters, but also on a number of external factors over which we have no real control. These external factors include, but are not restricted to, the current load of the machine and the operating system. Because of this, for fixed values of the tuning parameters we might get different values of the performance function even when performing the same task.

Even if we were able to fully isolate performance variation due to external factors, trying to find a minimum point in an arbitrary (and unknown) curve would require an exhaustive search of the entire space of values evaluating performance at each point. If the number of different values of each bundle is big this brute force approach is not feasible. Hence, we had to come up with heuristics to solve the problem. While the goal is to get the best performance possible, we were mostly interested in avoiding those k-tuples for which the performance was particularly bad. We have set this goal based on our experience in using the interface with a few test applications (including a database engine and parallel search algorithm). We found that there are frequently many points near the optimal point and that there is also often

another region where the application performance is abysmal. Thus, trying to get into the good region even if we don't find the absolute best point achieves most of the benefit of finding the optimal solution.

We had several other goals for our minimization algorithm: 1) it should not require too many evaluations of the performance function and 2) it should avoid first and second order derivatives. The algorithm that we developed is based on the simplex method for a finding a function's minimization [10].

The algorithm makes use of a simplex, which is a geometrical figure defined by k+1 connected points in a k-dimensions space. For the 2-dimensions space, the simplex is a triangle, and for the 3-d space the simplex is a non-degenerated tetrahedron. The Nelder-Mead simplex method approximates the extreme of a function by considering the worst point of the simplex and forming its symmetrical image through the center of the opposite (hyper) face. At each step a better point, making the simplex move towards the extreme, replaces the worst point. In our case the algorithm slips down the valley towards the minimum.

The algorithm described above assumes a well-defined function and works in a continuous space. However, neither of these assumptions holds in our situation. Thus we had to come up with a method to adapt the algorithm to deal with this. Rather than modifying the algorithm to deal with this problem, we simply used the resulting values from the nearest integer point in the space to approximate the performance at the selected point in the continuous space.

6. Experimental Results

We conducted series of experiments to evaluate the design and its performance. We first present results for the library specification layer. Then we use the Harmony server to tune the selected library through iterations to improve the overall performance.

6.1 Algorithm Tuning Experiments

We evaluate the library specification layer by first applying it to a simple data structure abstraction, and then apply it two commonly used math libraries. All of our tests were run on Redhat Linux with kernel 2.4.0 [17] on a Pentium-III 667MHz with 384 MB main memory. The metric used for all four test cases are the time to complete the requests.

Table Abstraction: The first set of libraries consists of two libraries. Each of them implements a two dimensional array. The two dimensional array is used to store data elements similar to a table. The focus of this test case is the ability to select different data structure based on memory access patterns. Two program libraries are implemented using linked lists and arrays. Each of them has its own advantages over the other one: link list takes less memory space for storage but longer time for insert, delete, and search operations; array takes more memory for storage but more efficient in insert, delete, and search operation. The full paper includes detailed results that show that the system can use memory vs. speed criteria to select an appropriate implementation of the table abstraction. Proper selection can reduce runtime by a factor of 20 or more and space by two-orders of magnitude for a set of randomly generated requests to store and lookup items in a table. The full paper also includes a study of the implementation of two compression algorithms that use a table abstraction. By Harmonizing the table, we can optimize the compression algorithms for either space or time.

Matrix Inversion: The second set of program libraries consists of two matrix inversion routines from LAPACK [3]. The major characteristic of the matrix is a Boolean indicating if the matrix is triangular. If the matrix is triangular, using the dedicated triangular matrix inversion library will have better performance. Otherwise, general matrix inversion library must be used. The result is shown in Figure 2; the library compares the triangular matrix by applying it to both the dedicated triangular inversion matrix library and the general matrix inversion matrix at the beginning. Later for each request, the library specification layer detects whether the request matrix is triangular and if so, the library specification layer will invokes the matrix inversion library optimized for triangular matrices. Otherwise, the library specification layer will just apply it to the general matrix inversion library.

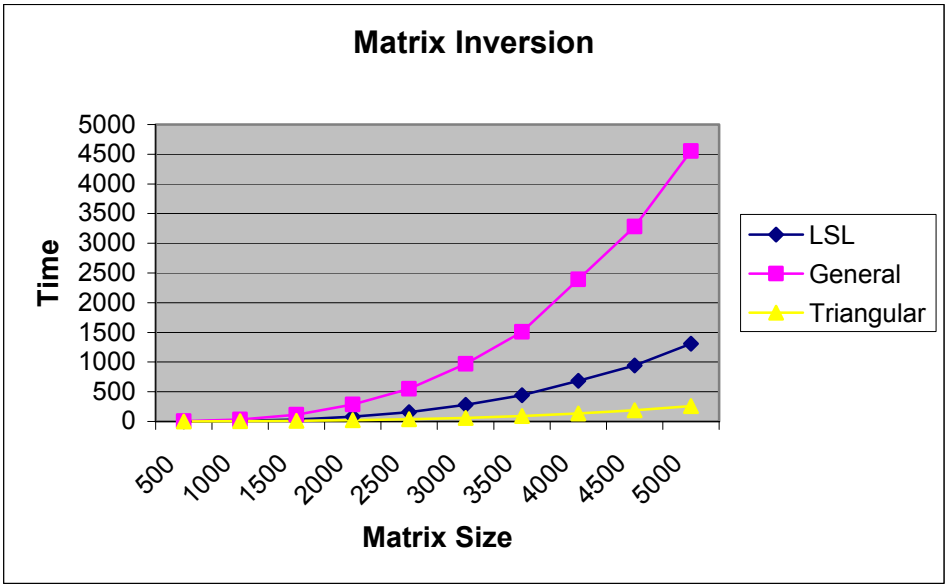

Figure 2*: Matrix Inversion Test Case*

6.2 Parameter Tuning Experiments

Once the library specification layer selects the underlying program library, the Harmony server tunes both the application and the library to improve the performance. Due to the complexity of measuring real applications and possible anomalies due to execution, we initially evaluated our search heuristic using several well-defined functions.

We tested the effectiveness of our search algorithm on three benchmark functions. The three functions were: a sombrero hat function, a quadratic function, and Rosenbrock's parabolic valley. The full paper includes results that show in 8-30 steps, our search algorithm can find a solution that is within 0.05% of the minimal value.

6.2.1 I/O Intensive Application

To evaluate our optimization system using a real application, we selected a 3-d volume reconstruction application [2] built on top of the Active Data Repository (ADR) middleware [9]. The 3-d volume reconstruction application uses digital images of a space to reconstruct the objects that are visible from the various camera angles. The ADR is an infrastructure that integrates storage, retrieval and processing of large multi-dimensional data sets. ADR provides the user with operations including index generation, data retrieval, scheduling across parallel machines, and memory management. The data is accessed through range queries (i.e., extract all data within a specified region of space). A range query is processed in two steps: query planning followed by query execution. As part of query execution, input and output items are mapped between coordinate systems and the data is aggregated to generate the final result. During the processing phase a temporary dataset, called the accumulator, is created to hold the results of the query being processed.

Because ADR is middleware used to build multiple applications, including the Harmony calls in the ADR code makes every application built on top of ADR tunable. The parameters we used were:

tileSize represents the size of the memory tile that is used by the ADR back-end to store information before the information is aggregated. It is the size of the tiles in which the above-mentioned accumulator will be partitioned if it does not fit into memory. This parameter has great influence on the query planning and query execution phases since it is somewhat analogous to the bock size in a computational code that has been blocked (tiled) to fit into a cache.

lowWatermark is the upper bound of the number of pending reads and number of ready reads that were issued to the disk in order to resolve a certain query.

maxReads is the maximum number of reads issued in order to resolve the current query if the number of pending read operations and the number of ready read operations are below the lowWatermark.

The original version of the volume reconstruction application used values for the above-described parameters that were determined by the ADR designers. To harmonize the application, we added calls to expose these parameters to the system. The environment in which we ran the experiments was a Linux cluster of 16 machines, each having two 450 MHz processors connected by 100 Mbps Ethernet.

To see how the Active Harmony infrastructure improves the running time of the Volume Reconstruction application we created a random set of queries that were submitted to the ADR back-end. First we ran them using the original version of the ADR and then the harmonized version. Figure 3 below presents the improvement we obtained in the processing time of each of the queries.

The Active Harmony system sped up query processing by up to 50% for the set of 70 random queries that we generated. However, the average improvement was about 10%. This is due to the fact that some of the queries that had some of the greatest speed-ups were very short, compared with others for which the improvement was less than 10%. The performance improvement for the longest query was about 18%, which for a query that took about 10 minutes to be completed reduced the running time by 2 minutes.

Since we don't know the shape of the performance curve (and thus what the best value is), another set of experiments was conducted that compares the behavior of the Active Harmony adaptation system to the brute force search for the best parameter values. For the purpose of the exhaustive search we submitted to the ADR back-end the same query for each tuple of parameter values. We then recorded the value of the performance function for each of the 1680 tuples. To test the behavior of the Active Harmony we submitted the same query 2000 times to the ADR back-end.

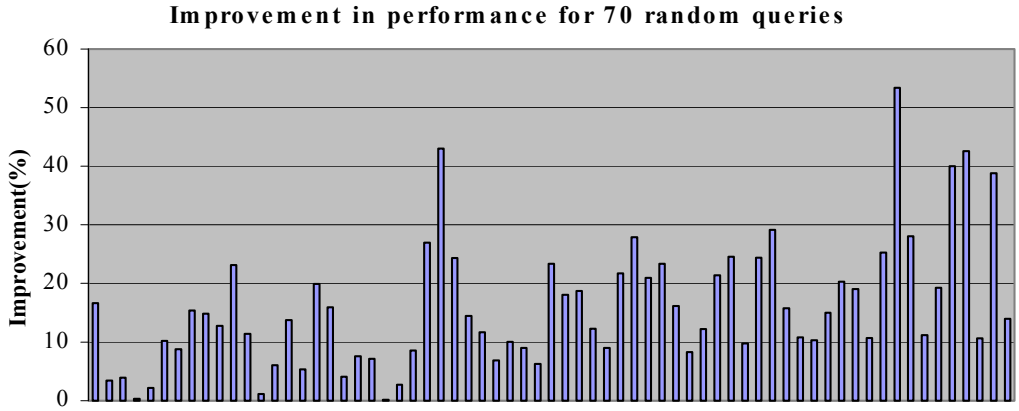

Figure 3: Performance improvement for the volume reconstruction application.

The brute force algorithm recorded values for the performance function of up to 25% slower than the optimum, while the range of values explored by the Active Harmony system was within 5% of the minimum. The minimum was reached by our system by exploring only 11 tuples (out of the almost 1,700 different possible tuples). Figure 4 below presents these results. The axes of the graph are as follows: the vertical one represents the performance function, while the horizontal ones are the tileSize and the lowWatermark. The values obtained for different values of the third parameter: maxReads are stacked one on top of the other in the graph. The lighter points in the graph are from the exhaustive search and are spread on the entire value space. The darker points (lower left corner) trace the path followed by the tuning algorithm and as it can be easily noticed from the graph, they are concentrated towards the point of minimum.

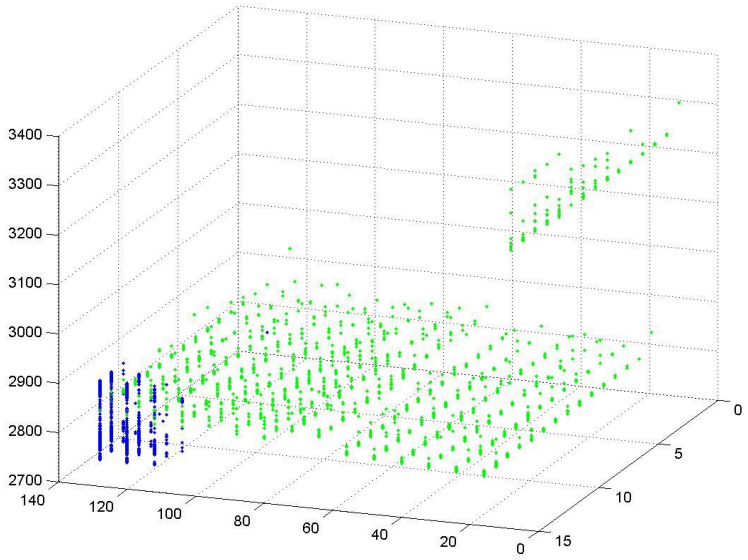

Figure 4: Performance curve (via exhaustive search) for the volume reconstruction application.

7. Related Work

There are several projects that have been seeking to develop techniques to allow applications to be responsive to their available resources or to allow them to be tuned at runtime. Computational Steering [5][6][11][12][13][14] provides a way for users to alter the behavior of an application under execution. Harmony's approach is similar in that applications provide hooks to allow their execution to be changed. Many computational steering systems are designed to allow the application semantics to be altered; for example, adding a particle to a simulation, as part of a problem-solving environment, rather than for performance tuning. Also, most computational steering systems are manual in that a user is expected to make the changes to the program.

One exception to this is Autopilot [13][14], which allows applications to be adapted in an automated way. Sensors extract quantitative and qualitative performance data from executing applications, and provide requisite data for decision-making. Autopilot uses a fuzzy logic to automate the decision making process. Their actuators execute the decision by changing parameter values of applications or resource management policies of underlying system. Harmony differs from Autopilot in that it tries to coordinate the use of resources by multiple libraries and applications.

The ATLAS [15] project has developed automatically tuned linear algebra libraries. They develop a methodology for the automatic generation of high efficient basic linear algebra routine for today's microprocessor. By using a code generator that probes and searches the system for an optimal set of parameters, it can produce highly optimized matrix multiply for a wide range of architectures. The difference between our work and ATLAS is that our work focuses on general applications that use program libraries rather than that of a specific library.

Another approach is application level scheduling. AppLeS [1] allows applications to be informed of the variations in resources and presented with candidate lists of resources to use. In this system, applications are informed of resource changes and provided with a list of available resource sets. Then, each application allocates the resources based upon a customized scheduling to maximize its own performance. The Network Weather Service [16] is used to forecast the network performance and available CPU percentage to AppLeS agents. Harmony differs from AppLes in that we try to optimize resource allocation between multiple libraries and applications, whereas AppLes lets each application or library adapt itself independently. In addition, by providing a structured interface for applications to disclose their specific

preferences, Harmony will encourage programmers to think about their needs in terms of options and their characteristics rather than as selecting from specific resource alternatives described by the system.

Prior work in the active Harmony project [7][8] concentrated on the API to make applications tunable, and in defining an interface to express the different options via a Resource Specification Language. This paper extends that work be providing an improved search algorithm (rather than a simple greedy approach). In addition, we describe the new Algorithm Adaptation layer that provides the glue code to allow existing (slightly) different APIs to be "harmonized."

8. Conclusion

This paper presented an infrastructure for tuning distributed applications for better performance and an optimization algorithm based on the simplex method for function minimization.

Based on a simple architecture and with minimal changes to the source code of the applications, Active Harmony provides the user the ability to improve the performance of an application using an automatic search through algorithms or parameters at runtime. Another significant advantage provided by the Active Harmony system is the ability to make applications sensitive to the external factors and parameters that characterize the environment in which they are executed. The results presented above demonstrate that Active Harmony can bring significant improvements to distributed applications and even more, it opens new horizons in adapting applications to dynamic environments.

References

[1] F. Berman and R. Wolski, "Scheduling from the Perspective of the Application," *Proceedings of the 5th International Symposium on High Performance Distributed Computing (HPDC-5)*.

[2] E. Borovikov, A. Sussman, and L. Davis, "An Efficient System for Multi-Perspective Imaging and Volumetric Shape Analysis," *Workshop on Parallel and Distributed Computing in Image Processing, Video Processing Multimedia (PDIVM'20001)*. April 2001, IEEE Computer Society Press.

[3] J. Dongarra et. al, "LAPACK – Linear Algebra PACKage," http://www.netlib.org/lapack/

[4] Jan Foster and Carl Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," published by *Morgan Kaufmann, July 1998*

[5] A. G. Geist, J. A. Kohl, and P. M. Papadopoulos, "CUMULVS: Providing Fault tolerance, Visualization, and Steering of Parallel Applications," *International Journal of Supercomputer Applications and High Performance Computing*, **11**(3), 1997, pp. 224-35.

[6] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, J. Vetter, and N. Mallavurupu, "Falcon: On-line Monitoring and Steering of Large-Scale Parallel Programs," *Frontiers '95*. Feb 6-9, 1995, McLean, VA, IEEE Press, pp. 422-429.

[7] J. K. Hollingsworth and P. J. Keleher, "Prediction and Adaptation in Active Harmony," *Cluster Computing*, 2 (1999), pp. 195-205

[8] P. J. Keleher, J. K. Hollingsworth, and D. Perkovic, "Exposing Application Alternatives," *ICDCS*. June 1999, Austin, TX, pp. 384-392.

[9] T. Kurc, C. Chang, R. Ferreira, A. Sussman, and J. Saltz, "Querying Very Large Multi-dimensional Datasets in ADR," *Proceedings of SC99*. Nov. 1999, Orlando, FL, ACM Press.

[10] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *Computer J.*, **7**(4), 1965, pp. 308--313.

[11] S. G. Parker and C. R. Johnson, "SCIRun: a scientific programming environment for computational steering," *Supercomputing'95*. Nov. 1995, San Diego, Vol. II, pp. 1419-39.

[12] Daniel A. Reed, Christopher L. Elford, Tara. M. Madhyastha, Evgenia Smirni, and Stephen E. Lamm. "The next frontier: interactive and closed loop performance steering," *Proc. 1996 ICPP Workshop on Challenges for Parallel Process*. (Bloomingdale, Ill., Aug. 1996) 20-31 (1996).

[13] Randy L. Ribler, Huseyin Simitci, and Daniel A. Reed, "The Autopilot Performance-Directed Adaptive Control System," *Future Generation Computer Systems, special issue (Performance Data Mining)*, Volume 18, Number 1, 2001, pp. 175-187.

[14] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: Adaptive Control of Distributed Applications," *High Performance Distributed Computing*. Aug. 1998, Chicago, IL, pp. 172-9.

[15] R. Clint Whaley and Jack J. Dongarra, "Automatically tuned linear algebra software (ATLAS)," *In Proceedings of SC'98*, Orlando, FL, November 1998.

[16] R. Wolski, "Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service," *High Performance Distributed Computing (HPDC)*. August 1997, Portland, Oregon, IEEE Press, pp. 316-325

[17] The Linux Kernel Archives. http://www.kernel.org/