

A Framework for Adaptive Runtime Management of AMR Applications*

Sumir Chandra, Shweta Sinha, Manish Parashar
The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
Piscataway, NJ 08854, USA
{sumir,shwetas,parashar}@caip.rutgers.edu

Yeliang Zhang, Jingmei Yang, Salim Hariri
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, USA
{zhang,jm_yang,hariri}@ece.arizona.edu

Abstract

This paper presents a runtime management framework for adaptive mesh refinement applications that is capable of reactively and proactively managing and optimizing application execution using current system and application state, predictive models for system behavior and application performance, and an agent based control network. The overall goal of this research is to enable large-scale dynamically adaptive scientific and engineering simulations on distributed, heterogeneous and dynamic execution environments such as the computational “grid”. The design, prototype implementation, and evaluation of the framework using a distributed and adaptive Richtmyer-Meshkov CFD kernel is presented.

Keywords: Adaptive runtime management; Structured adaptive mesh refinement; Dynamic applications; Heterogeneous distributed computing; Performance characterization.

1 Introduction

Next-generation scientific and engineering simulations of complex physical phenomena will be built on widely distributed, highly heterogeneous and dynamic, networked computational “grids”. These simulations will provide new insights into complex systems such as interacting black holes and neutron stars, formations of galaxies, subsurface flows in oil reservoirs and aquifers, and dynamic response of materials to detonation. However, configuring and managing the execution of these applications to exploit the underlying computational power in spite of its heterogeneity and dynamism, presents many challenges. The overall goal of this research is to realize an adaptive runtime framework capable of reactively and proactively managing and optimizing application execution using current system and application state, predictive models for system behavior and application performance, and an agent based control network. Its overarching motivation is enabling very large-scale, dynamically adaptive scientific and engineering simulations on distributed, heterogeneous and dynamic execution environments such as the computational “grid”.

This paper presents the design, prototype implementation, and evaluation of a proactive and reactive system-sensitive runtime management framework for Adaptive Mesh Refinement (AMR) applications. System capabilities and current state are obtained using the NWS (Network Weather Service) [13] resource monitoring tool and used to appropriately distribute and load balance the dynamic AMR computation domain. Performance prediction functions hierarchically combine analytical, experimental and empirical performance models to predict the performance of the application, and to determine when the overheads of dynamic load balancing are justified and if it is

*The work presented here was supported by the National Science Foundation via grants numbers ACI 9984357 (CA-REERS), EIA 0103674 (NGS) and by DOE ASCI/ASAP (Caltech) via grant number PC295251.

beneficial to redistribute load. An active control network combines sensors, actuators and application management agents and provides the mechanism to adapt the application at runtime.

2 Problem Description: Enabling Realistic Simulations using Adaptive Mesh Refinement

The design of the adaptive runtime framework is driven by specific problems in enabling realistic simulations using AMR techniques. In this paper, we use the Richtmyer-Meshkov (RM) instability encountered in compressible fluid dynamics (RM3D application)¹. The RM instability occurs when a plane shock interacts with a corrugated interface between two fluids of different densities. As a result of such an interaction, interface perturbation starts to grow because the transmitted shock is converging at the wave peak and diverging at the valley. Converging shock increases pressure and accelerates perturbation peak into the second fluid. RM instabilities occur over a wide range of scales, from nearly microscopic objects, such as laser fusion pellets, to objects of astronomical size, such as supernovae.

A key challenge in such a simulation is that the physics exhibits multiple scales of length and time. If one were to employ zoning, which resolves the smallest scales, the required number of computational zones would be prohibitive. One solution is to use Adaptive Mesh Refinement (AMR) with multiple independent timesteps, which allows the grid resolution to adapt to a local estimate of the error in the solution. With AMR, the number of zones along with their location in the problem space is continuously changing. Besides dynamic communication and storage requirements, another challenge is that the local physics may change significantly from zone to zone as fronts move through the system. The heterogeneous and dynamic load per zone is a problem for load-balancing schemes, which typically assign blocks of physically adjacent zones to a single processor. Furthermore, the resource requirements of each zone in the simulation have to be predicted.

Distributed implementations of these simulations thus lead to interesting challenges in dynamic resource allocation, data-distribution and load balancing, communications and coordination, and resource management. As a result, key requirements for an adaptive runtime management framework include *Dynamic Partitioning Support*, *Adaptive Communication Support*, and *Dynamic Application Configuration Support*. However, the complexity and heterogeneity of the environment make selection of a “best” match between system resources, application algorithms, problem decompositions, mappings and load distributions, communication mechanisms, etc., non-trivial. System dynamics coupled with application adaptivity makes application configuration and runtime management a significant challenge. In this paper, we address dynamic system-sensitive partitioning and load-balancing.

3 Adaptive Runtime Management Framework: Design Overview

The runtime management framework is composed of three key components: a system characterization and abstraction component, a performance analysis module, and an active control network module, discussed as follows.

3.1 System Characterization and Abstraction

The objective of the system characterization/abstraction component is to monitor, abstract and characterize the current state of the underlying computational environment, and use this information to drive the predictive performance functions and models that can estimate its performance in the near future. Networked computational environments such as the computational “grid” are highly dynamic in nature. Thus, it is imperative that the application management system be able to react to this dynamism and make runtime decisions to ensure that the application’s requirements are satisfied and its performance optimized. These decisions include selecting

¹RM3D has been developed by Ravi Samtaney as part of the virtual test facility developed at the ASCI/ASAP Center at the California Institute of Technology (<http://www.cacr.caltech.edu/ASAP/>)

the appropriate number, type, and configuration of the computing elements, appropriate distribution and load-balancing schemes, the most efficient communication mechanism, as well as the right algorithms and parameters at the application level. Furthermore, proactive application management by predicting system behavior will enable a new generation of applications that can tolerate the dynamics of the grid and truly exploit its computational capabilities. The system characterization component in our framework builds on existing frameworks, such as NWS [13], that have been deployed in networked environments and are capable of monitoring and reporting the current status of individual computational, network, and storage elements.

3.2 Performance Analysis Module

The performance analysis module is built on *Performance Functions*. Performance Functions (PF) describe the behavior of a system component, subsystem or compound system in terms of changes in one or more of its attributes. For example, processor instructions (arithmetic, logical or control) are normally characterized by the number of clock cycles required to execute them. Using the PF concept, we can characterize the operations and performance of any resource in a distributed environment. Once the PFs of each resource used by an application are defined, we compose these PFs to generate an overall end-to-end PF that characterizes and quantifies application performance. Note that the computation of overall PF function is analogous to the calculation of end-to-end block transfer function from individual block transfer functions in control system theory.

Our PF-based modeling approach includes three steps. First, we identify the attributes that can accurately express and quantify the operation and performance of a resource (e.g., Clock speed, Error, Capacity). The second step is to use experimental and analytical techniques to obtain the PF that characterizes and quantifies the performance of each system component in terms of these attributes. The final step is to compose the component PFs to generate an overall PF that can be used during runtime to estimate and project the operation and performance of the application for any system and network state. This composition approach is based on the performance interpreting approach for parallel and distributed applications [8, 9].

3.3 Active Control Network

The underlying mechanisms for adaptive runtime management of AMR applications are realized by an active control network of sensors, actuators, and management agents. This network overlays the application data-network and allows application components to be interrogated, configured, and deployed at runtime to continually ensure that application requirements are satisfied in the most efficient manner. In addition to the management agents, sensors and actuators need to be embedded within the application and/or system software as they define the interfaces and implement the mechanisms for adaptation. Furthermore, these sensors and actuators can be directly deployed (and co-located) with the application's computational data structures in a straightforward manner. This approach has been successfully used to embed and deploy sensors and actuators for interactive computational steering of large, distributed and adaptive applications, both, by us [5] and others [12, 10, 2].

3.4 Adaptive Application Management

The key goal of the runtime management framework is to develop policies and mechanisms for both "application sensitive" and "system sensitive" runtime adaptations of AMR applications. The former is based on current application state while the latter is driven by current system state and system performance predictions. Application sensitive adaptations [1] use the current state of the application to drive the runtime adaptations. The abstraction and characterization of the application state is used to drive the resource allocation, partitioning and mapping of application components onto the grid, selection and configuration of partitioning and load-balancing algorithms, communication mechanisms, etc. System sensitive application management [11] uses current and predicted system state characterization to make application adaptation decisions. For example, the information about the current

load and available memory will determine the granularity of the mapping of application components to processing nodes, while available communication bandwidths will determine the communication strategy to be used. Similarly, application level algorithms may be selected based on the type, specifications, and status of the underlying architecture. Finally, the availability and “health” of computing elements on the grid may determine the nature (refined grid size, aspect ratios, etc.) of refinements to be allowed.

4 Experimental Evaluation of the Runtime Management Framework

We have developed and deployed a prototype runtime management framework that uses current system state and predictive performance functions to proactively and reactively manage the distribution and load-balancing of AMR applications. The prototype framework has been integrated into the GrACE (Grid Adaptive Computational Engine) infrastructure’s [6, 7] adaptive runtime system. GrACE is a data-management framework for parallel/distributed AMR, and is being used to provide AMR support for varied applications including reservoir simulations, computational fluid dynamics, seismic modeling, and numerical relativity. This section presents the implementation and experimental evaluation of this prototype using the RM3D CFD kernel.

4.1 Reactive System Sensitive Partitioning and Load Balancing

The adaptive runtime framework reacts to system capabilities and current system state to select and tune distribution parameters by dynamically partitioning and load balancing the AMR grid hierarchies. Current system state is obtained at runtime using the NWS resource monitoring tool. System state information along with system capabilities are then used to compute relative computational capacities of each of the computational nodes. These relative capacities are used by the “system-sensitive” partitioner for dynamic distribution and load-balancing. NWS periodically monitors and dynamically forecasts the performance delivered by the various network and computational resources over a given time interval. Measurements include the fraction of CPU time available for new processes, the fraction of CPU available to a process that is already running, end-to-end TCP network latency, end-to-end TCP network bandwidth, free memory, and the amount of space unused on a disk.

This system information provided by NWS is used to compute a relative capacity metric for each processor as follows [11]. Let us assume that there are K processors in the system among which the partitioner distributes the workload. For node k , let \mathcal{P}_k be the percentage of CPU available, \mathcal{M}_k the available memory, and \mathcal{B}_k the link bandwidth. The available resource at k is first converted to a fraction of total available resources, i.e.

$$P_k = \mathcal{P}_k / \sum_{i=1}^K \mathcal{P}_i, M_k = \mathcal{M}_k / \sum_{i=1}^K \mathcal{M}_i, \text{ and } B_k = \mathcal{B}_k / \sum_{i=1}^K \mathcal{B}_i \quad (1)$$

The relative capacity C_k of a processor is then defined as the weighted sum of these normalized quantities

$$C_k = w_p P_k + w_m M_k + w_b B_k \quad (2)$$

where w_p , w_m , and w_b are the weights associated with the relative CPU, memory, and link bandwidth availabilities, respectively, such that $w_p + w_m + w_b = 1$. The weights are application dependent and reflect its computational, memory, and communication requirements. Note that

$$\sum_{k=1}^K C_k = 1 \quad (3)$$

If the total work to be assigned to all the processors is denoted by L , then the work L_k assigned to the k th processor can be computed as $L_k = C_k L$. The overall operation is shown in Figure 1.

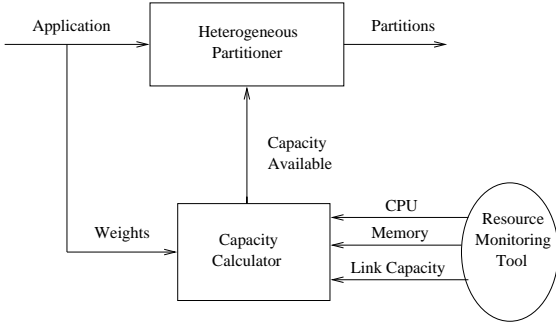


Figure 1. System sensitive adaptive AMR partitioning

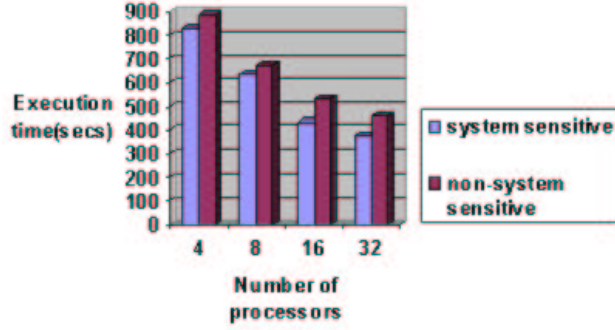


Figure 2. Application execution time for the system sensitive and default (non-system sensitive) partitioning schemes

Number of Processors	Percentage Improvement
4	7%
8	6%
16	18%
32	18%

Table 1. Improvement due to system-sensitive adaptive partitioning.

The system sensitive adaptive partitioner is evaluated using the RM3D CFD kernel on a Linux-based workstation cluster. The kernel used 3 levels of factor 2 space-time refinements on a base mesh of size 128*32*32. The cluster consisted of 32 nodes interconnected by fast Ethernet (100MB). The experimental setup consisted of a synthetic load generator (for simulating heterogeneous loads on the cluster nodes) and an external resource monitoring system (i.e. NWS). The evaluation comprised of comparing the runtimes and load-balance generated for the system sensitive partitioner with those for the default partitioning scheme provided by GrACE. This latter scheme assumes homogeneous processors and performs an equal distribution of the workload on the processors.

The application execution time using the system sensitive partitioner and the default non-system sensitive partitioner is plotted in Figure 2 and the improvement achieved is listed in Table 1. System sensitive partitioning reduced execution time by about 18% in the case of 32 nodes. We believe that the improvement will be more significant in the case of a cluster with greater heterogeneity and load dynamics.

The adaptivity of the system sensitive partitioner to system dynamics are evaluated for a cluster with 4 nodes, with the relative capacities $C_1, C_2, C_3,$ and C_4 computed as 16%, 19%, 31%, and 34% respectively. The three system characteristics, viz. CPU, memory, and link bandwidth, are assumed to be equally important, i.e. $w_p = w_m = w_b = 1/3$, and the application regrids every 5 iterations. The load assignment for the GrACE default and the system sensitive (*ACEHeterogeneous*) partitioners are plotted in Figures 3 and 4 respectively.

The percentage of load imbalance for the default and the system-sensitive partitioning schemes is plotted in Figure 5. For the k th processor, the load imbalance I_k is defined as

$$I_k = \frac{|W_k - L_k|}{L_k} \times 100 \quad \% \quad (4)$$

As expected, the GrACE default partitioner generates large load imbalances as it does not consider relative

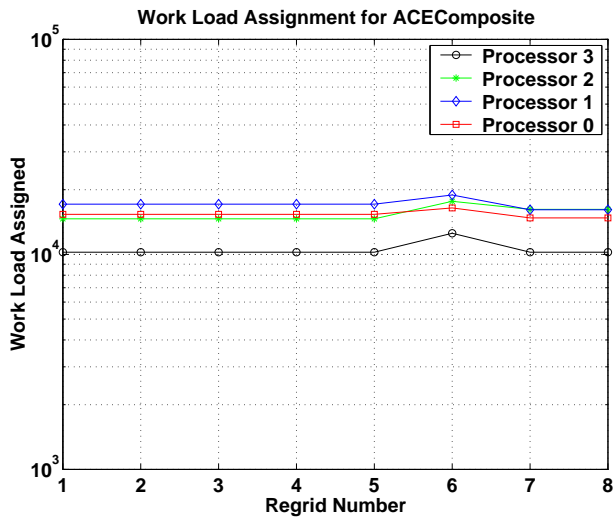


Figure 3. Workload assignments for default partitioning scheme (Relative capacities of processors 0 – 3 are 16%, 19%, 31%, 34%)

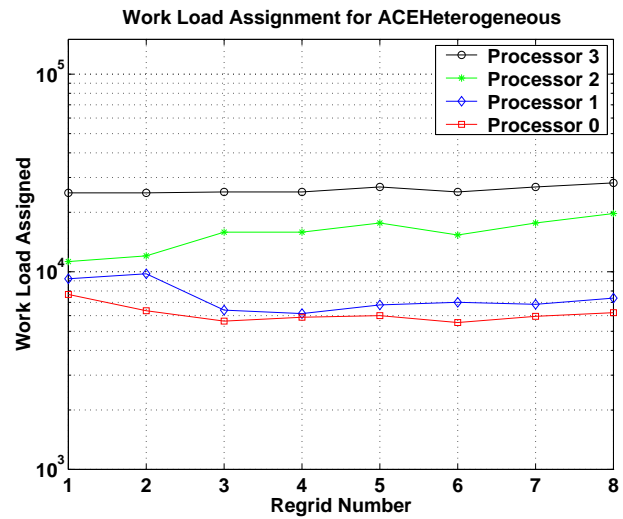


Figure 4. Workload assignments for ACEHeterogeneous partitioning scheme (Relative capacities of processors 0 – 3 are 16%, 19%, 31%, 34%)

capacities. The system sensitive partitioner produces about 45% smaller imbalances. Note that the load imbalances in the case of the system sensitive partitioner are due to the constraints (minimum box size and aspect ratio) that have to be satisfied while breaking boxes.

In order to evaluate the ability of the system-sensitive partitioner to adapt to load dynamics in the cluster, the synthetic load generator was used on two processors to dynamically vary the system load. The load assignments at each processor were computed for different sensing frequencies. Table 2 illustrates the effect of sensing frequency on overall application performance. Dynamic runtime sensing improves application performance by as much as 45% compared to sensing only once at the beginning of the simulation. Figure 6 shows the relative processor capacities and load assignments for a sensing frequency of 20 iterations. The frequency of sensing depends on the load dynamics and can affect application performance. In our experimental setup, the best application performance was achieved for a sensing frequency of 20 iterations.

Number of Processors	Execution time with Dynamic Sensing (secs)	Execution time with Sensing only once (secs)
2	423.7	805.5
4	292.0	450.0
6	272.0	442.0
8	225.0	430.0

Table 2. Comparison of execution times using static (only once) and dynamic sensing (every 40 iterations).

4.2 Proactive Management using Performance Functions

The adaptive runtime framework uses performance prediction functions to estimate application execution times and to determine when the benefits of dynamic load redistribution exceed the costs of repartitioning and data movement. The performance functions (PF) model the execution of the AMR-based RM3D application and describe

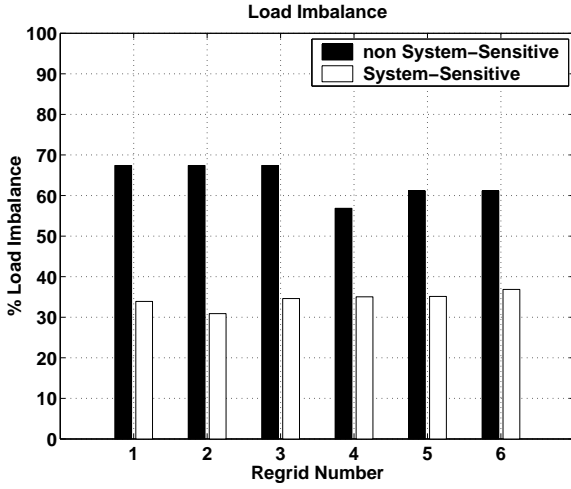


Figure 5. Load imbalance for the system sensitive and default partitioners

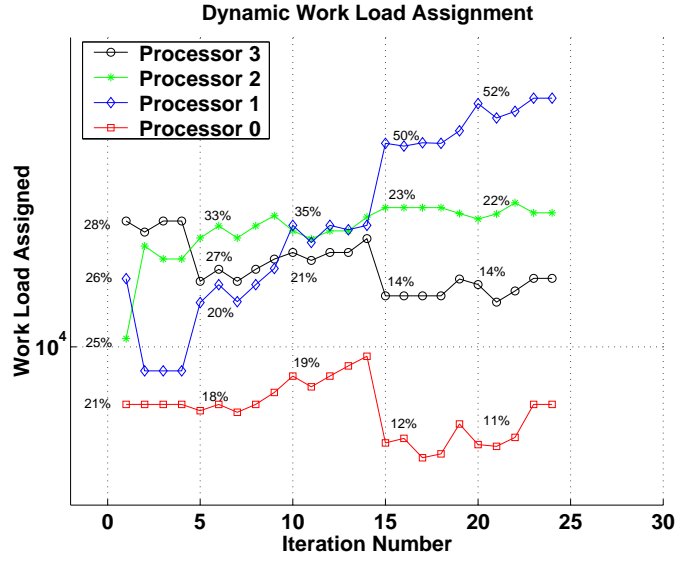


Figure 6. Dynamic load allocation for a system state sensing frequency of 20 iterations

its overall behavior with respect to the desired metric. In this experiment, we use the computational load as the metric and model application execution time with respect this attribute. The processing time for each application component on the machine of choice (IBM SP and Linux Beowulf, in our case) is measured in terms of the load, and the measurements are fed to a neural network to obtain the corresponding PF.

IBM SP “Seaborg”²: For our evaluation, we obtain the following two PFs: PF_s denotes the PF associated with small loads ($\leq 30,000$ work units), and PF_h denotes the PF associated with high loads ($> 30,000$ work units). The PF increase linearly as the number of processes on each processor increases. The PFs for small and high loads are as follows:

$$PF_s = \sum_{i=0}^{10} a_i * x^i \quad (5)$$

$$PF_h = \sum_{i=0}^9 a_i * x^i \quad (6)$$

where, a_i ($i=0,1,\dots,10$) are constants and x is the computational load. The coefficients for the PFs for small and large loads are listed in Table 3.

Linux Beowulf “Discover”³: Our evaluation yields a single PF on the Linux cluster, as follows:

$$PF = \sum_{i=0}^{10} b_i * x^i \quad (7)$$

where, b_i ($i=0,1,\dots,10$) are coefficients (listed in Table 4) and x is the workload.

²The National Energy Research Scientific Computing Center (NERSC) IBM SP RS/6000, named seaborg.nersc.gov, is a distributed memory parallel supercomputer with 2,944 compute processors among 184 compute nodes

³Discover is a 16-node Beowulf cluster at Rutgers University

PF_s small load	a_0	0.24819702	a_3	2.6207385e-10	a_6	-3.3198527e-22	a_9	5.1340974e-36
	a_1	0.001067243	a_4	-4.6855919e-14	a_7	1.3903117e-26	a_{10}	-3.1716301e-41
	a_2	-7.9638733e-07	a_5	4.9943821e-18	a_8	-3.56914e-31		
PF_h high load	a_0	-670.06183	a_3	-1.0509561e-10	a_6	-8.3594889e-24	a_9	7.3129295e-39
	a_1	0.064439362	a_4	2.5819559e-15	a_7	1.7978927e-28		
	a_2	-7.5567587e-07	a_5	1.3903143e-19	a_8	-1.8236343e-33		

Table 3. Constant Coefficients for Performance Functions on IBM SP “Seaborg”.

PF	b_0	6.0826507	b_3	2.4465223e-13	b_6	-3.6882935e-29	b_9	7.8471222e-47
	b_1	0.00048341426	b_4	-2.10405e-18	b_7	7.8747669e-35	b_{10}	-2.5582951e-53
	b_2	-1.5930319e-08	b_5	1.1051294e-23	b_8	-1.0441739e-40		

Table 4. Constant Coefficients for Performance Functions on Linux Beowulf “Discover”.

Tables 5 and 6 show that the error incurred in modeling the execution time based on the PF modeling approach is low, roughly between 0-8% for the IBM SP and between 0-6% for the Beowulf cluster. Details about the PF-based approach for modeling large-scale distributed systems are found in [3, 4].

The PF-based model is evaluated on the Beowulf cluster to determine the effect of dynamic load balancing on application recompose time. Load redistribution is expensive for small load variations and is justified when the workload imbalance exceeds a certain threshold. The RM3D application ran on 8 processors for redistribution thresholds of 0 and 1, and used 3 levels of factor 2 refinements on a base mesh of size $64*16*16$. Recompose time for threshold of 1 improved by almost 100% as compared to when a regular, periodic load redistribution (threshold of 0) is used.

5 Conclusions

In this paper, we presented the design, prototype implementation, and evaluation of an adaptive runtime framework capable of reactively and proactively managing and optimizing application execution using current system and application state, predictive models for system behavior and application performance, and an active control network. The overarching motivation for this research is to enable large-scale dynamically adaptive scientific and engineering simulations on distributed, heterogeneous and dynamic execution environments such as the computational “grid”. System capabilities and current state are obtained using the NWS resource monitoring tool and used to appropriately distribute the dynamic AMR computation domain. Performance functions are used to predict application performance and to determine when it is beneficial to redistribute load. An active control net-

Workload (grid units)	Actual time (seconds)	PF derived time (seconds)	Error rate (%)
13824	0.442005	0.408765	7.5
18432	0.514049	0.508943	0.99
23040	0.573535	0.562287	1.9
35072	0.708781	0.708931	0

Table 5. Accuracy of Performance Functions on IBM SP “Seaborg”.

Workload (grid units)	Actual time (seconds)	PF derived time (seconds)	Error rate (%)
74880	9.48232	9.98521	5.3
97344	9.8688	9.88585	0.17
123656	10.3557	9.76117	5.74
173056	11.0533	10.4147	5.77
274560	11.2683	11.2393	0.26
430496	14.9336	15.5478	4.11

Table 6. Accuracy of Performance Functions on Linux Beowulf “Discover”.

work consisting of application sensors, actuators and application management agents adapts application execution at runtime to optimize performance. Experimental results using a distributed and adaptive Richtmyer-Meshkov CFD kernel are presented. We are currently extending this evaluation to larger systems and more heterogeneous configurations and to different application domains.

References

- [1] S. Chandra, J. Steensland, M. Parashar, and J. Cummings. An Experimental Study of Adaptive Application Sensitive Partitioning Strategies for SAMR Applications. *2nd LACSI Symposium*, Oct. 2001.
- [2] T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, and J. Subhlok. ReMoS: A Resource Monitoring System for Network Aware Applications, Technical Report, CMU-CS-97-194. CMU, 1997.
- [3] S. Hariri and et al. A Hierarchical Analysis Approach for High Performance Computing and Communication Applications. *Hawaii Intl. Conference on System Sciences*, 1999.
- [4] S. Hariri, H. Xu, and A. Balamash. A Multilevel Modeling and Analysis of Network-Centric Systems. Special Issue of *Microprocessors and Microsystems Journal*, Elsevier Sci. on Engineering Complex Computer Systems, 1999.
- [5] S. Kaur, V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. Engineering a Distributed Computational Collaboratory. *34th Hawaii Intl. Conference on System Sciences*, Jan. 2001.
- [6] M. Parashar and J. C. Browne. System Engineering for High Performance Computing Software: The HDDA/DAGH Infrastructure for Implementation of Parallel Structured Adaptive Mesh Refinement. *IMA Volume 117: Structured Adaptive Mesh Refinement Grid Methods*, IMA Volumes in Mathematics and its Applications. Springer-Verlag, pp. 1-18, 2000.
- [7] M. Parashar and J. C. Browne. On Partitioning Dynamic Adaptive Grid Hierarchies. Proceedings of the *29th Annual Hawaii International Conference on System Sciences*, January 1996.
- [8] M. Parashar and S. Hariri. Interpretive Performance Prediction for Parallel Application Development. *J. of Parallel and Distributed Computing*, vol. 60(1), 17-47, Jan. 2000.
- [9] M. Parashar and S. Hariri. Compile-Time Performance Interpretation of HPF/Fortran 90D. *IEEE Parallel and Distributed Technology*, Spring, 1996.
- [10] R. Ribler, J. Vetter, H. Simitci, and D. Reed. Autopilot: Adaptive Control of Distributed Applications. *7th IEEE Intl. Symp. on High-Performance Distributed Computing*, July 1998.
- [11] S. Sinha and M. Parashar. Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. *3rd IEEE Intl. Conference on Cluster Computing*, 435-442, 2001.
- [12] J.S. Vetter and K. Schwan. Optimizations for language-directed computational steering. *IPPS 99*, 1999.
- [13] R. Wolski. Forecasting Network Performance to Support Dynamic Scheduling using the Network Weather Service. *6th IEEE Symp. on HPDC*, 1997.