Adaptive Runtime Management of SAMR Applications*

Sumir Chandra[1], Shweta Sinha[1], Manish Parashar[1],
Yeliang Zhang[2], Jingmei Yang[2], and Salim Hariri[2]

[1] The Applied Software Systems Laboratory
Rutgers, The State University of New Jersey
Piscataway, NJ 08854, USA
{sumir, shwetas, parashar}@caip.rutgers.edu
[2] Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, USA
{zhang, jm_yang, hariri}@ece.arizona.edu

Abstract. This paper presents the design, prototype implementation, and evaluation of a runtime management framework for structured adaptive mesh refinement applications. The framework 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".
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

* Support for this work was provided by the NSF via grants numbers ACI 9984357 (CAREERS), EIA 0103674 (NGS) and EIA-0120934 (ITR), DOE ASCI/ASAP (Caltech) via grant number PC295251, and the DOE Scientific Discovery through Advanced Computing (SciDAC) program via grant number DE-FC02-01ER41184.

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 Structured Adaptive Mesh Refinement (SAMR) applications. System capabilities and current state are obtained using the NWS (Network Weather Service) [9] 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 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.

The research presented in this paper extends our prior work on system-sensitive runtime management and integrates the different runtime approaches (reactive system-sensitive partitioning and proactive management using performance functions) within a single adaptive and automated framework.

2 Enabling Realistic Simulations Using AMR

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 3-D Richtmyer-Meshkov (RM3D[3]) instability encountered in compressible fluid dynamics. 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.

[3] RM3D has been developed by Ravi Samtaney as part of the virtual test facility at the Caltech ASCI/ASAP Center (http://www.cacr.caltech.edu/ASAP).

Distributed implementations of these simulations lead to interesting challenges in dynamic resource allocation, data-distribution and load balancing, communications and coordination, and resource management. Furthermore, the complexity and heterogeneity of the environment make the selection of a "best" match between system resources, application algorithms, problem decompositions, mappings and load distributions, communication mechanisms, etc., nontrivial. 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 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, described 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 satisfy application requirements and optimize performance. These decisions include selecting 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.

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. 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.

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 interpretation approach for parallel and distributed applications [6, 7].

3.3 Active Control Network

The underlying mechanisms for adaptive runtime management of SAMR 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 ensure that application requirements are satisfied. Sensors and actuators are embedded within the application and/or system software and define interfaces and mechanisms for adaptation. This approach has been successfully used to embed and deploy sensors and actuators for interactive computational steering of large, distributed and adaptive applications [4].

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 SAMR 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 [8] 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 Runtime Management Framework: Evaluation

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 SAMR applications. The prototype framework has been integrated into the GrACE (Grid Adaptive Computational Engine) [5] infrastructure's 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 SAMR 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 [8]. 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.

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

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

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

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 $\sum_{k=1}^{K} C_k = 1$. If L is the total work to be assigned to all the processors, then the work L_k assigned to the kth processor can be computed as $L_k = C_k L$. The overall operation is shown in Figure 1.

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

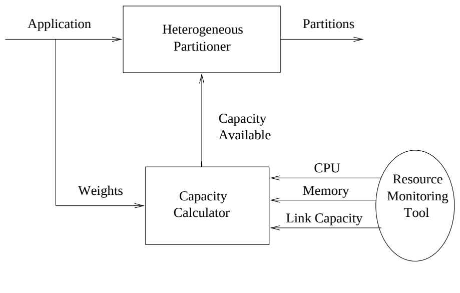

Fig. 1. System sensitive adaptive AMR partitioning

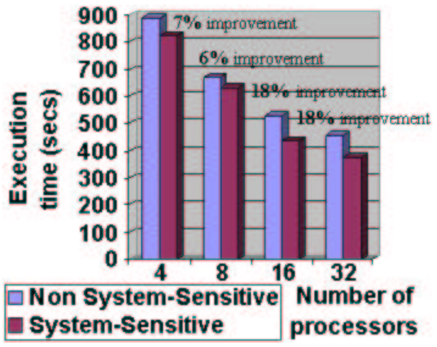

Fig. 2. Improvement in execution time due to system sensitive partitioning

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 improvement in application execution time using the system sensitive partitioner as compared to the default non-system sensitive partitioner is illustrated in Figure 2. 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. For the kth processor, the load imbalance I_k is defined as

$$I_k = \frac{|W_k - L_k|}{L_k} \times 100 \quad \% \tag{3}$$

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

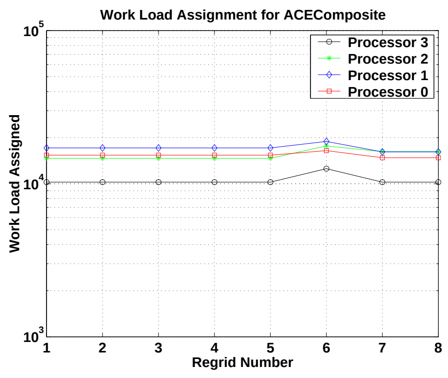

Fig. 3. Workload assignments for default partitioning scheme (Relative capacities of processors $0 - 3$ are 16%, 19%, 31%, 34%)

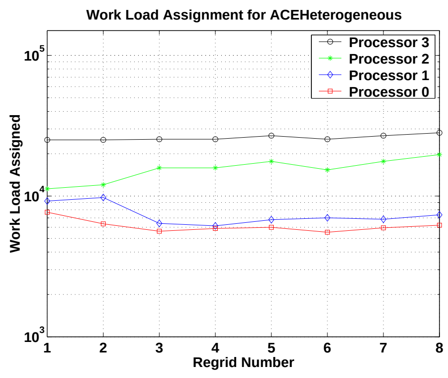

Fig. 4. Workload assignments for ACE-Heterogeneous partitioning scheme (Relative capacities of processors $0-3$ are 16%, 19%, 31%, 34%)

processors to dynamically vary the system load. The load assignments at each processor were computed for different sensing frequencies. Table 1 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 5 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.

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 SAMR-based RM3D application and describe 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 to 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 then used to obtain the corresponding PF.

IBM SP "Seaborg"[4]: For our evaluation, we obtain the following two PFs: PF_s denotes the PF associated with small loads ($\leq 30,000$ work units) and

[4] The 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.

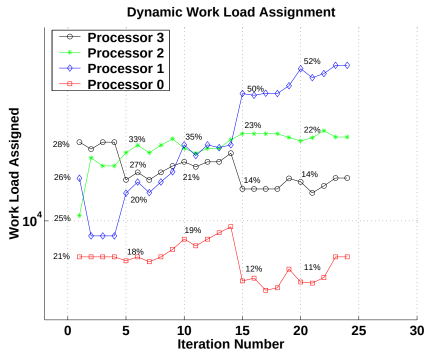

Fig. 5. Dynamic load allocation for a system state sensing frequency of 20 iterations

Table 1. Comparison of execution times using static (only once) and dynamic sensing (every 40 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

PF_h denotes the PF associated with high loads ($> 30,000$ work units). The PFs increase linearly as the number of processes on each processor increases. The PFs for small and high loads are as follows:

$$PF_s = \sum_{i=0}^{10} a_i * x^i \quad \text{and} \quad PF_h = \sum_{i=0}^{9} a_i * x^i \tag{4}$$

where a_i (i=0,1,...,10) are constants and x is the computational load. The coefficients for the PFs for small and large loads are listed in Table 2.

Table 2. Constant coefficients for performance functions on IBM SP "Seaborg"

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

Linux Beowulf "Discover"[5]: The performance function evaluation on the Linux cluster yields a single PF as follows:

$$PF = \sum_{i=0}^{10} b_i * x^i \tag{5}$$

where b_i (i=0,1,...,10) are coefficients (listed in Table 3) and x is the workload.

[5] Discover is a 16-node Beowulf cluster at Rutgers University.

Table 3. Constant coefficients for performance function on Linux Beowulf "Discover"

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		

Tables 4 and 5 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 [2, 3].

Table 4. Accuracy of performance functions on IBM SP "Seaborg"

Workload (grid units)	Actual time (sec)	PF derived time (sec)	Error rate (%)
13824	0.4420	0.4088	7.5
18432	0.5140	0.5089	0.99
23040	0.5735	0.5623	1.9
35072	0.7088	0.7089	0

Table 5. Accuracy of performance functions on Linux Beowulf "Discover"

Workload (grid units)	Actual time (sec)	PF derived time (sec)	Error rate (%)
74880	9.4823	9.9852	5.3
97344	9.8688	9.8859	0.17
123656	10.3557	9.7612	5.74
173056	11.0533	10.4147	5.77
274560	11.2683	11.2393	0.26
430496	14.9336	15.5478	4.11

This PF-based model is used by the adaptive framework to determine when the benefits of dynamic load redistribution exceed the costs of repartitioning and data movement. For N processors in the system, let $GlbLoad$ denote the global workload for the structured dynamic grid hierarchy and $LocLoad_k$ denote the local load for processor k. The ideal workload per processor is given by $IdlLoad = GlbLoad/N$. Using the PF-based approach, execution time estimates are obtained for ideal and local workloads for each processor, denoted by $PFtimeIdl$ and $PFtimeLoc$ respectively. Load redistribution is typically expensive for small load variations; however, it is justified when the workload imbalance exceeds a certain threshold, defined by

$$Thresh = \frac{PFtimeIdl - PFtimeLoc}{PFtimeLoc} \tag{6}$$

A threshold of 0 indicates regular periodic load redistribution regardless of the load-balancing costs. A high threshold represents the ability of the application hierarchy to tolerate workload imbalance and determines when the overheads of dynamic load balancing are justified and if it is beneficial to redistribute load.

The RM3D evaluation on the Beowulf cluster analyzes the effect of dynamic load balancing on application recompose time in order to achieve better performance. The experimental setup consists of the RM3D application executing on 8 processors for redistribution thresholds of 0 and 1. The application uses 3 levels of factor 2 space-time refinements on a base mesh of size 64*16*16 with regriding every 4 time-steps. Threshold of 1 considers the costs and benefits of redistributing load and results in recompose time being reduced by half (improvement of almost 100%) as compared to when a 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". 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. Proceedings of the *2nd Los Alamos Computer Science Institute Symposium* (also best research poster at *Supercomputing Conference 2001*), October 2001.
2. S. Hariri and et al. A Hierarchical Analysis Approach for High Performance Computing and Communication Applications. Proceedings of the *32nd Hawaii International Conference on System Sciences*, 1999.
3. S. Hariri, H. Xu, and A. Balamash. A Multilevel Modeling and Analysis of Network-Centric Systems. Special Issue of *Microprocessors and Microsystems Journal*, Elsevier Science on Engineering Complex Computer Systems, 1999.
4. S. Kaur, V. Mann, V. Matossian, R. Muralidhar, and M. Parashar. Engineering a Distributed Computational Collaboratory. Proceedings of the *34th Hawaii International Conference on System Sciences*, January 2001.
5. M. Parashar and J. Browne. On Partitioning Dynamic Adaptive Grid Hierarchies. Proceedings of the *29th Hawaii International Conference on System Sciences*, January 1996.
6. M. Parashar and S. Hariri. Interpretive Performance Prediction for Parallel Application Development. *Journal of Parallel and Distributed Computing*, vol. 60(1), pp. 17-47, January 2000.
7. M. Parashar and S. Hariri. Compile-Time Performance Interpretation of HPF/Fortran 90D. *IEEE Parallel and Distributed Technology*, Spring 1996.
8. S. Sinha and M. Parashar. Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. Proceedings of the *3rd IEEE International Conference on Cluster Computing*, pp. 435-442, 2001.
9. R. Wolski. Forecasting Network Performance to Support Dynamic Scheduling using the Network Weather Service. Proceedings of the *6th IEEE Symposium on High Performance Distributed Computing*, 1997.