Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters*

Shweta Sinha and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering
Rutgers, The State University of New Jersey
94 Brett Road, Piscataway, NJ 08855-8060
{*shwetas, parashar*}*@caip.rutgers.edu*

Abstract

This paper presents the design and evaluation of an adaptive, system sensitive partitioning and load balancing framework for distributed structured adaptive mesh refinement applications on heterogeneous and dynamic cluster environments. The framework uses system capabilities and current system state to select and tune appropriate partitioning parameters (e.g. partitioning granularity, load per processor) to maximize overall application performance.

Keywords: *System-sensitive adaptive partitioning, dynamic load-balancing, heterogeneous computing, structured adaptive mesh refinement.*

1 Introduction

Dynamically adaptive methods for the solution of partial differential equations that employ locally optimal approximations can yield highly advantageous ratios for cost/accuracy when compared to methods based upon static uniform approximations. These techniques seek to improve the accuracy of the solution by dynamically refining the computational grid in regions of high local solution error. Distributed implementations of these adaptive methods offer the potential for the accurate solution of realistic models of important physical phenomena. These implementations however, lead to interesting challenges in dynamic resource allocation, dynamic data-distribution and load balancing, communications and coordination, and resource management.

Moving these applications to dynamic and heterogeneous cluster computing environments introduces a new level of complexity. These environments require the runtime selection and configuration of application components and parameters based on the availability and state of the resources. However, the complexity and heterogeneity of the environment make selection of a "best" match between system resources, mappings and load distributions, communication mechanisms, etc., non-trivial. System dynamics coupled with application adaptivity makes application and run-time management a significant challenge.

In this paper we present the design and evaluation of an adaptive, system sensitive partitioning and load balancing framework for distributed structured adaptive mesh refinement (AMR) applications on heterogeneous and dynamic cluster environments. The framework uses system capabilities and current system state to select and tune appropriate distribution parameters. Current system state is obtained using the NWS (Network Weather Service) [3] resource monitoring tool. The current system state along with system capabilities are then used to compute the relative computational capacities of each of the computational nodes in the cluster. These relative capacities are used by a heterogeneous "system-sensitive" partitioner for dynamic distribution and load-balancing. The heterogeneous partitioner has been integrated into the GrACE (Grid Adaptive Computational Engine) infrastructure [1, 2] 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.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes the adaptive grid structure defined by hierarchical adaptive mesh-refinement techniques. Section 4 introduces the GrACE adaptive computational engine. Section 5 outlines the archi-

*The research presented in this paper is based upon work supprted by the National Science Foundation under Grant Numbers ACI 998357 (CAREERS) and WU-HT-99-19 P029876F (KDI) awarded to Manish Parashar.

Reference	Environment State	Application State
S. M. Figueira, et. al	static	static
J. Watts, et. al	dynamic	static
M. Maheswaran, et. al	dynamic	static
R. Wolski, et. al	dynamic	static

Table 1. *Summary of the related work in dynamic partitioning techniques for heterogenous cluster.*

tecture of the system sensitive runtime management framework. Section 6 describes the framework implementation and presents an experimental evaluation. Section 7 presents some concluding remarks.

2 Related Work

There exists a large amount of research addressing load balancing techniques for heterogeneous clusters. Most of these works however assume that the applications and the system is static. Some research has also addressed the mapping of *static* applications to *dynamic* heterogeneous environments, where the capabilities of the environments change. The discussion presented here focuses on approaches that target heterogeneous systems. Figueira et. al in [5] have defined an algorithm which determines the best allocation of tasks to resources in heterogeneous systems. It however does not take into account the dynamism in the environment and uses a static application. Watts et. al in [6] have also presented techniques for dynamic load balancing in heterogeneous computing environments. They have shown that performance improves when the computers capacities' are calculated dynamically. Maheswaran et. al in [7] have proposed a new dynamic algorithm, called the hybrid remapper, for improving initial static matching and scheduling. The hybrid-remapper uses the run-time values that become available for subtask completion times and machine availabilities during application execution time. They have also used a static application. Wolski et. al in [3] talk about finding a suitable match in a heterogeneous network when the network conditions are changing. Related work in dynamic partitioning techniques for heterogeneous clusters is summarized in Table 1.

The system sensitive partitioning/load-balancing framework presented in this paper addresses both, dynamic heterogeneous environments and dynamically adaptive applications.

3 Problem Description: Distributed Structured AMR Applications

Dynamically adaptive numerical techniques for solving differential equations provide a means for concentrating computational effort to appropriate regions in the computational domain. In the case of hierarchical structured AMR methods, this is achieved by tracking regions in the domain that require additional resolution and dynamically overlaying finer grids over these regions. Structured AMR-based techniques start with a base coarse grid with minimum acceptable resolution that covers the entire computational domain. As the solution progresses, regions in the domain requiring additional resolution are tagged and finer grids are overlaid on the tagged regions of the coarse grid. Refinement proceeds recursively so that regions on the finer grid requiring more resolution are similarly tagged and even finer grids are overlaid on these regions. The resulting grid structure is a dynamic adaptive grid hierarchy. The adaptive grid hierarchy corresponding to the AMR formulation by Berger & Oliger [4] is shown in Figure 1, and the associated operations are outlined below.

Time Integration: Time integration is the update operation performed on each grid at each level of the adaptive grid hierarchy. Integration uses an application specific difference operator.

Inter-Grid Operations: Inter-grid operations are used to communicate solutions values along the adaptive grid hierarchy. Two primary inter-grid operations are *Prolongations*, defined from a coarser grid to a finer grid, and *Restrictions*, defined from a finer grid to a coarser grid.

Regriding: The regriding operation consists of three steps: (1) flagging regions needing refinement based on an application specific error criterion, (2) clustering flagged points, and (3) generating the refined grid(s). The regriding operation can result in the creation of a new level of refinement or additional grids at existing levels, and/or the deletion of existing grids.

3.1 Decomposing the Adaptive Grid Hierarchy

Key requirements for a decomposition scheme used to partition the adaptive grid hierarchy across processors are: (1) expose available data-parallelism; (2) minimize communication overheads by maintaining inter-level and intra-level locality; (3) balance overall load distribution; and (4) enable dynamic load re-distribution with minimum overheads. A balanced load distribution and efficient redistribution is particularly critical for parallel AMR-based applications as different levels of the grid hierarchy have different computational loads. In case of the Berger-Oliger

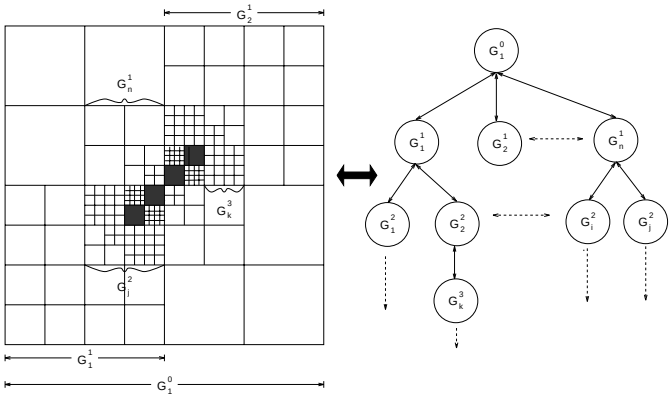

Figure 1. *Adaptive Grid Hierarchy - 2D (Berger-Oliger AMR Scheme)*

AMR scheme for time-dependent applications, space-time refinement result in refined grids which not only have a larger number of grid elements but are also updated more frequently (i.e. take smaller time steps). The coarser grid are generally more extensive and hence its computational load cannot be ignored. Furthermore, the AMR grid hierarchy is a dynamic structure and changes as the solution progresses, thereby making efficient dynamic re-distribution critical.

4 Grid Adaptive Computational Engine (GrACE)

The adaptive "system sensitive" partitioning mechanisms presented in this paper have been integrated into the GrACE [1, 2] adaptive runtime system. GrACE is an object-oriented toolkit for the development of parallel and distributed applications based on a family of adaptive mesh-refinement and multigrid techniques. It provides a high-level programming abstraction and allows users to simply and directly express distributed computations on adaptive grids, and is built on a distributed dynamic data-management substrate.

4.1 GrACE Distributed Dynamic Data Management Substrate

The GrACE substrate implements a "semantically specialized" distributed shared memory and provides distributed and dynamic data-management support for large-scale parallel adaptive applications. The lowest layer of the infrastructure implements a Hierarchical Distributed Dynamic Array (HDDA). The HDDA provides array semantics to hierarchical and physically distributed data. HDDA objects encapsulate dynamic load-balancing, interactions and communications, and consistency management. The next layer adds application semantics to HDDA objects to implement application objects such as grids, meshes and trees. This layer provides an object-oriented programming interface for directly expressing multi-scale, multi-resolution AMR computations. The upper layers of the infrastructure provide abstractions, components and modules for method-specific computations.

4.1.1 Hierarchical Distributed Dynamic Array

The primitive data structure provided by the data-management substrate is an array which is hierarchical in that each element of the array can recursively be an array, and dynamic in that the array can grow and shrink at runtime. The array of objects is partitioned and distributed across multiple address spaces with communication, synchronization and consistency transparently managed for the user. The lowest level of the array hierarchy is an object of arbitrary size and structure. The primary motivation for defining such a generalized array data-structure is that most application domain algorithms are formulated as operations on grids and their implementation is defined as operations on arrays. Such array based formulations have proven to be simple, intuitive and efficient, and are extensively optimized by current compilers. Providing an array interface to the dynamic data-structures allows implementations of new parallel and adaptive algorithms to reuse existing kernels at each level of the HDDA hierarchy. Like conventional arrays HDDA translates index locality (corresponding spatial application locality) to storage locality, and must maintain this locality despite its dynamics and distribution. The HDDA implementations is composed of a hierarchical index space and distributed dynamic storage and access mechanisms. The former is derived directly from the application domain using space-filling mappings [8], and the latter uses extensible hashing techniques [9]. Figure 2 shows the overall HDDA storage scheme. More information about the GrACE data-management substrate and the HDDA can be found in [1]

5 System Sensitive Runtime Management Architecture

A block diagram of the adaptive, system sensitive runtime partitioning framework is shown in Figure 3. The framework consists of three key components - the system state monitoring tool, the capacity calculator and the system sensitive partitioner. In this framework, we first monitor the state of resources associated with the different computing nodes in the cluster, and use this information to compute their relative computational capacities. The relative capacities are then used by the heterogeneous system-sensitive

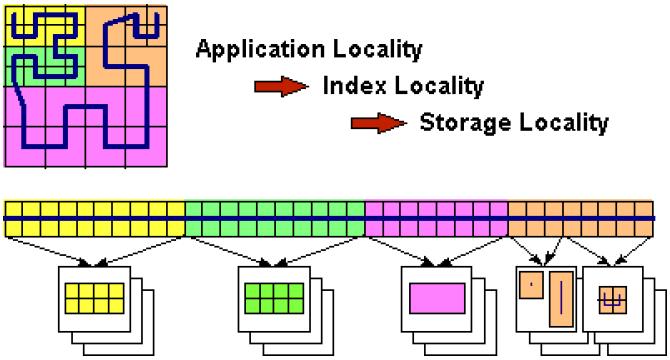

Figure 2. *Distributed Storage of Dynamic Objects*

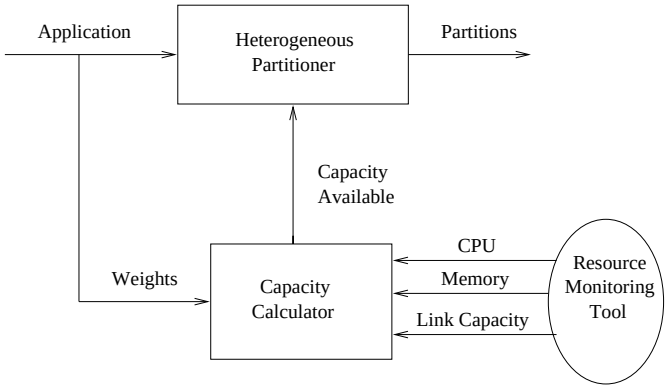

Figure 3. *Architecture of the System Sensitive Runtime Management System*

partitioner for dynamic distribution and load-balancing. The different components are described below.

5.1 Resource Monitoring Tool

System characteristics and current state are determined at run-time using an external resource monitoring tool. The resource monitoring tool gathers information about the CPU availability, memory usage and link-capacity of each processor. This information is then passed to the Capacity Calculator as shown in Figure 3.

5.2 Capacity Metric

Using system information obtained from the resource monitoring tool, a relative capacity metric is computed for each processor using a linear model as follows. Let us assume that there are K processors in the system among which the partitioner distributes the work load. 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, as provided by NWS. The available resource at k is first converted to a fraction of total available resources, i.e. $P_k = \mathcal{P}_k / \sum_{i=1}^{K} \mathcal{P}_i$, $M_k = \mathcal{M}_k / \sum_{i=1}^{K} \mathcal{M}_i$, and $B_k = \mathcal{B}_k / \sum_{i=1}^{K} \mathcal{B}_i$. The relative capacity C_k of a processor is then defined as the weighted sum of these normalized quantities

$$C_k = w_p P_k + w_m M_k + w_b B_k \quad (1)$$

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 the total work to be assigned to all the processors is denoted by L, then the work L_k that can be assigned to the kth processor can be computed as $L_k = C_k L$.

5.3 The System Sensitive Partitioner

The system sensitive partitioner, called *ACEHeterogeneous*, has been integrated into the GrACE runtime and provides adaptive partitioning and load-balancing support for AMR applications. In GrACE, component grids in the adaptive grid hierarchy are maintained as a list of bounding boxes. A bounding box is a rectilinear region in the computational domain and is defined by a lower bound, upper bound and a stride (defined by the level of refinement) along each dimension. Every time the applications regrids, the bounding box list is updated and is passed to the partitioner for distribution and load balancing. A high level description of the partitioning process is presented below.

- The relative capacities $C_k, k = 1, \ldots, K$ of the K processors over which the application is to be distributed and load balanced are obtained from the Capacity Calculator as shown in Figure 3.
- The total work L associated with the entire bounding box list is calculated.
- Using the capacity metric, the ideal work load L_k that can be assigned to the kth processor is computed.
- The bounding boxes are then assigned to the processors, with the kth processor receiving a total work load of W_k which is approximately equal to L_k.
- Both the list of bounding boxes as well as the relative capacities of the processors are sorted in an increasing order, with the smallest box being assigned to the processor with the smallest relative capacity. This eliminates unnecessary breaking of boxes (described below).
- If the work associated with a bounding box exceeds the work the processor can perform, the box is broken

into two in a way that the work associated with at least one of the two broken boxes is less than or equal to the work the processor can perform. While breaking a box the following constrains are enforced:

  - Minimum box size: All boxes much be greater than or equal to this size. As a consequence of enforcing this constraint, the total work load W_k that is assigned to processor k may differ from L_k thus leading to a "slight" load imbalance.
  - Aspect ratio: The *aspect ratio* of a bounding box, defined as the ratio of the longest side to the shortest side. To maintain a good aspect ratio, a box is always broken along the longest dimension.

- The local output list of bounding boxes is returned to GrACE which then allocates the work to the particular processor.

6 Experimental Evaluation

A 3D compressible turbulence kernel executing on a linux based workstation cluster is used to experimentally evaluate the adaptive, system sensitive partitioning framework. The kernel solved the Richtmyer-Meshkov instability, and used 3 levels of factor 2 refinement on a base mesh of size 128x32x32. The cluster consisted of 32 nodes interconnected by fast ethernet (100MB). The experimental setup consisted of a synthetic load generator and an external resource monitoring system. The evaluation consisted of comparing the runtimes and load-balance generated for the system sensitive partitioner with those for the default space-filling curve based partitioning scheme provided by GrACE. This latter scheme performs an equal distribution of the workload on the processors. The adaptivity of the system sensitive partitioner to system dynamics, and the overheads of sensing system state were also evaluated.

6.1 Experimental Setup

6.1.1 Synthetic Load Generation

In order to compare the two partitioning schemes, it is important to have an identical experimental setup for both cases. Hence, the experimentation was to be performed in a controlled environment so that the dynamics of the system state was the same in both cases. This was achieved using a synthetic load generator to load processors with artificial work. The load generator decreased the available memory and increased CPU load on a processor, thus lowering its capacity to do any work. The load generated on the processor increased linearly at a specified rate until it reached the desired load level. Note that multiple load generators were run on a processor to create interesting load dynamics.

6.1.2 Resource Monitoring

We used the Network Weather Service (NWS) [3] resource monitoring tool to obtain runtime information about system characteristics and current system state in our experiments. NWS is a distributed system that periodically monitors and dynamically forecasts the performance delivered by the various network and computational resources over a given time interval. In our experiments, we used NWS to monitor 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 bandwidth, and free memory. NWS has been engineered to be as non-intrusive as possible with a typical CPU utilization of less than 3% and a memory requirement of about 3300 KB [3].

6.1.3 System Sensitive Load Distribution

The following example illustrates the operation of the system sensitive partitioner. Consider a cluster with four nodes with the synthetic load generator used to load two of the nodes. Using the current system state provided by NWS, the capacity calculator computes the relative capacities C_1, C_2, C_3 and C_4 as approximately 16%, 19%, 31% and 34% respectively. In these calculations, the three system characteristics, viz. CPU, memory, and link bandwidth, were assumed to be equally important to the application, i.e. $w_p = w_m = w_b = 1/3$. The relative capacities are then fed to the *ACEHeterogeneous* partitioner, which uses them to partition the overall work load L among the four processors. In this case the four processors are assigned work loads of $L_1 = 0.16L$, $L_2 = 0.19L$, $L_3 = 0.30L$ and $L_4 = 0.34L$, respectively. The partitions appropriately assigns boxes (breaking large boxes if necessary) to processors to satisfy this distribution.

6.1.4 Dynamic Load Sensing

The system sensitive partitioner queries NWS at runtime to sense system load, computes the current relative capacities of the processors and distributes the workload based on these capacities. The sensing frequency depends on the dynamics of the cluster, and the overheads associated with querying NWS and computing relative capacities, and has to be chosen to balance these two factors. More frequent sensing (every few regrid steps) allows the system to adapt to rapid changes in system load but increases the sensing overheads. On the other hand, infrequent sensing (a few times during application execution) reduces these overheads but may increase overall execution time by preventing system from reacting to the cluster's load dynamics. Our experiments show that the overhead of probing NWS on a node, retrieving its system state, and computing its relative capacity is about 0.5 secs.

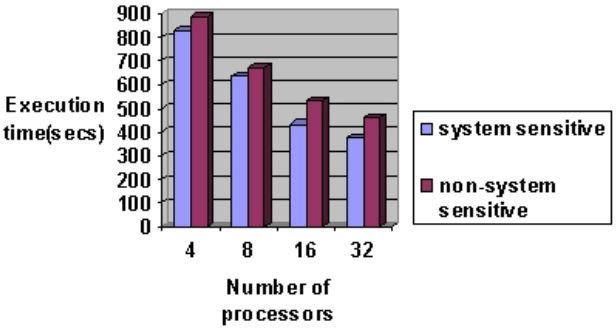

Figure 4. *Application execution time for the system sensitive and default (non-system sensitive) partitioning schemes*

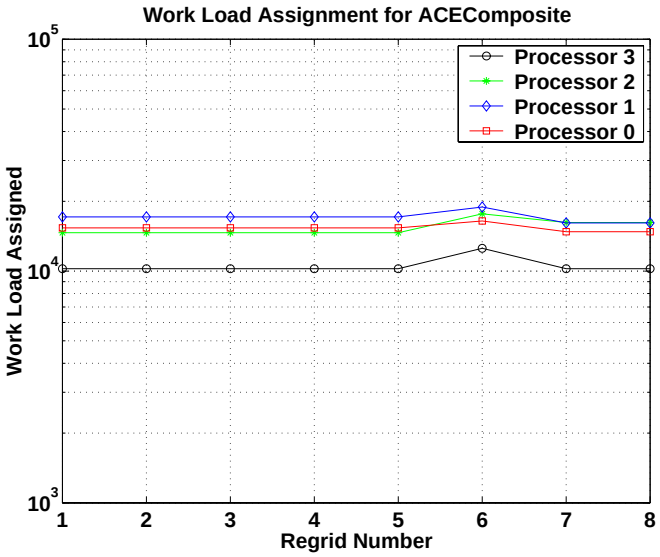

Figure 5. *Work load assignments for the default partitioning scheme (Relative capacities of processors $0-3$ equal to 16%, 19%, 31%, and 34% respectively).*

6.2 Experimental Results

6.2.1 Application performance improvement

The total application execution time using the system sensitive partitioner and the default non-system sensitive partitioner is plotted in Figure 4. In this experiment, the application was run under similar load conditions using the two partitioners. We calculate the relative capacities of the processors once before the start of the simulation. 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. Furthermore, tuning the sensing frequency also improves performance as shown in a later experiment.

6.2.2 Load balance achieved

This experiment investigates the load assignments and the effective load balance achieved using the two partitioners. In this experiment the relative capacities of the four processors were fixed at approximately 16%, 19%, 31% and 34%, and the application regrid every 5 iterations. The load assignment for the GrACE default and the system sensitive (*ACEHeterogeneous*) partitioners are plotted in Figures 5 and 6 respectively. As expected, the GrACE default partitioner attempts to assign equal work to each processor irrespective of its capacity. The system sensitive partitioner however assigns work based on each processor's relative capacity.

The percentage of load imbalance for the GrACE default and the system-sensitive partitioning schemes is plotted in Figure 7. For the kth processor, the load imbalance I_k is defined as

$$I_k = \frac{|W_k - L_k|}{L_k} \times 100 \quad \% \tag{2}$$

As expected, the GrACE default partitioner generates large load imbalances as it does not consider relative capacities. The system sensitive partitioner produces 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.

6.2.3 Adaptivity to Load Dynamics

This experiment evaluates the ability of the system sensitive partitioner to adapt to the load dynamics in the cluster, and the overheads involved in sensing the current state. In this experiment, the synthetic load generator was used on two of the processors to dynamically vary the system load. The load assignments at each processor was computed for different sensing frequencies. Figure 8 shows the load assignment in the case where NWS was queried once before the start of the application and two times during the application run, i.e. approximately every 45 iterations. The figure also shows the relative capacities of the processors at each sampling. It can be seen that as the load (and hence the relative capacities) of the processors change, the partitioning routine adapts to this change by distributing the work load accordingly. Also note that as the application adapts, the

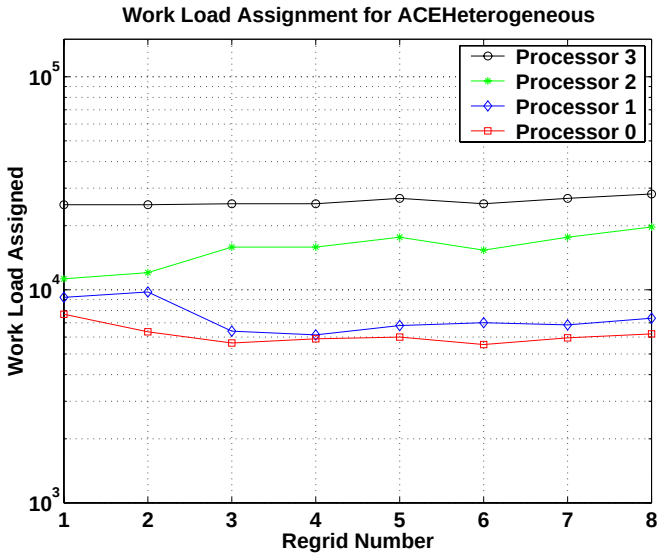

Figure 6. *Work-load assignments for the ACEHeterogeneous partitioning scheme (Relative capacities of processors $0-3$ equal to $16\%, 19\%, 31\%, and\, 34\%$ respectively).*

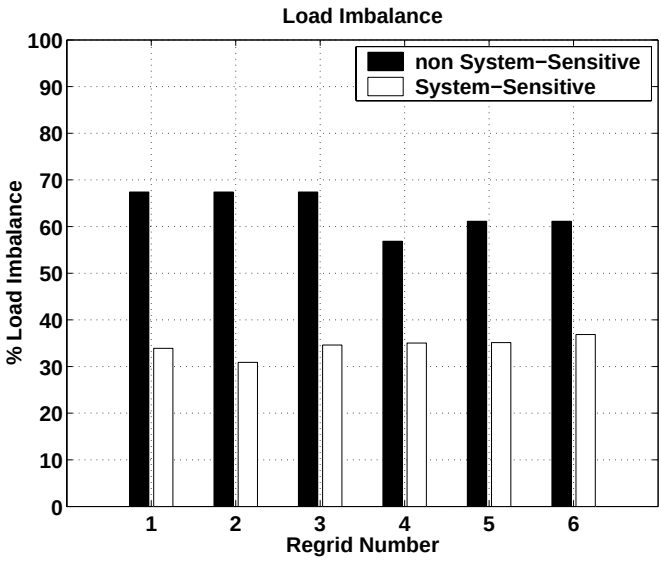

Figure 7. *Load imbalance for the system sensitive and default partitioners*

No. of Processors	Exec. time with Dynamic Sensing (secs)	Exec. 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 and dynamic sensing*

Frequency of calculating capacities	Execution time (secs)
10 iterations	316
20 iterations	277
30 iterations	286
40 iterations	293

Table 3. *Execution time for a four processor run when NWS is probed at different iterations*

total work load to be distributed varies from one iteration to the next. As a result, the work load assigned to a particular processor is different in different iterations even though the relative capacity of the processor does not change.

Tables 2 and 3 illustrate the effect of sensing frequency on overall application performance. The synthetic load dynamics are the same in each case. Table 2 compares the application execution times for the cases where the system state is queried only once at the beginning, and where it is queried every 45 iterations. It can be seen that dynamic runtime sensing significantly improves application performance. Table 3 presents application run time for different sensing frequencies - i.e. sensing every 10, 20, 30 and 40 iterations. The best application performance is achieved for a sensing frequency of 20 iteration. This number largely depends on the load dynamics of the cluster and the sensing overheads. Figure 9 shows the relative processor capacities and load assignments when system state is sensed every 20 iterations.

7 Conclusions

This paper presented a system sensitive partition and load-balancing framework for distributed AMR applications in heterogeneous cluster environments. The framework uses system capabilities and current system state to appropriately distribute the application and balance load. The framework has been implemented within the GrACE runtime system. System state is dynamically monitored using the Network Weather Service (NWS), and is used to compute current relative capacities of the computing nodes. These capacities are then used for partitioning and load balancing. An experimental evaluation of the framework was presented using a 3D AMR CFD kernel executing on a linux based cluster. A synthetic load generator was used to load the nodes in the cluster for the experiments. The exper-

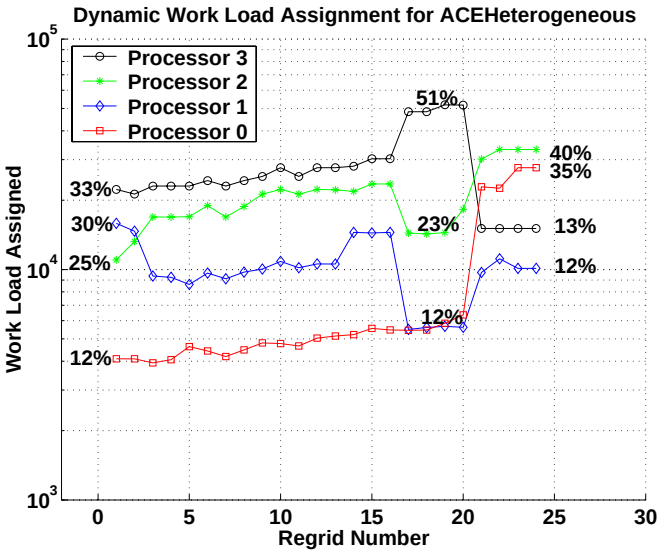

Figure 8. *Dynamic load allocation for a system state sensing frequency of 45 iterations*

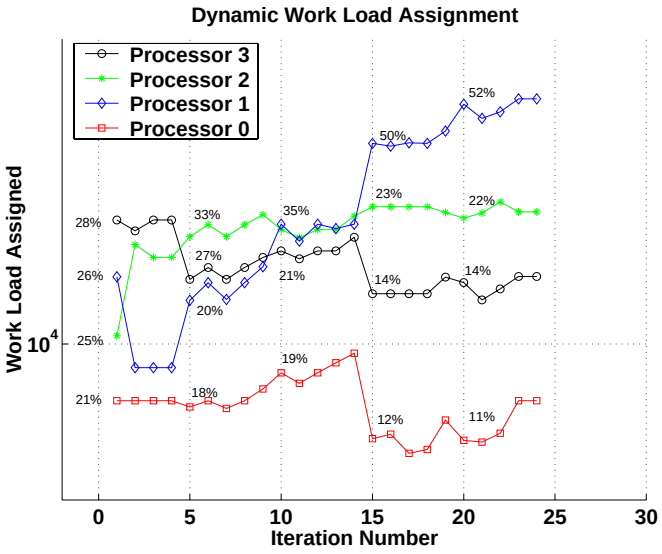

Figure 9. *Dynamic load allocation for a system state sensing frequency of 20 iterations*

iments showed that dynamic system sensitive partitioning can significantly improve the load balance achieved and overall application performance. Furthermore, application performance is sensitive to the frequency of sensing of the system state. This frequency has to be chosen based on the sensing overheads and the cluster load dynamics.

References

[1] 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", *in IMA Volume 117: Structured Adaptive Mesh Refinement Grid Methods, IMA Volumes in Mathematics and its Applications*. Springer-Verlag, pp. 1-18, 2000.

[2] Manish Parashar and James C. Browne, "*On Partitioning Dynamic Adaptive Grid Hierarchies*," Proceedings of the 29th Annual Hawaii International Conference on System Sciences, January, 1996.

[3] Rich Wolski, Neil T. Spring and Jim Hayes, "*The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*," Future Generation Computing Systems, 1999.

[4] Marsha J. Berger and Joseph Oliger "*Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*," Journal of Computational Physics, pp. 484-512, 1984.

[5] Silvia M. Figueira and Francine Berman "*Mapping Parallel Applications to Distributed Heterogeneous Systems, "* UCSD CS Tech Report # CS96-484, June 1996.

[6] Jerrel Watts, Marc Rieffel and Stephen Taylor "*Dynamic Management of Heterogeneous Resources*," High Performance Computing, 1998.

[7] M. Maheswaran and H. J. Seigel " *A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems*," 7th IEEE Heterogeneous Computing Workshop (HCW '98), Mar. 1998, pp. 57-69.

[8] H. Sagan, Space-filling curves, Springer-Verlag, 1994.

[9] R. Fagin, Extendible Hashing - A Fast Access Mechanism for Dynamic Files, *ACM TODS*, **4**:315–344, 1979.