Investigating Autonomic Runtime Management Strategies for SAMR Applications[⋆]

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

[1] Department of Electrical and Computer Engineering, Rutgers University, 94 Brett Road, Piscataway, NJ 08854, USA. Email: {sumir, parashar}@caip.rutgers.edu

[2] Department of Electrical and Computer Engineering, University of Arizona, 1230 E. Speedway, Tucson, AZ 85721, USA. Email: {jm_yang, zhang, hariri}@ece.arizona.edu

Dynamic structured adaptive mesh refinement (SAMR) techniques along with the emergence of the computational Grid offer the potential for realistic scientific and engineering simulations of complex physical phenomena. However, the inherent dynamic nature of SAMR applications coupled with the heterogeneity and dynamism of the underlying Grid environment present significant research challenges. This paper presents application/system sensitive reactive and proactive partitioning strategies that form a part of the GridARM autonomic runtime management framework. An evaluation using different SAMR kernels and system workloads is presented to demonstrate the improvement in overall application performance.

KEY WORDS: GridARM autonomic runtime management framework; structured adaptive mesh refinement; application/system sensitive reactive and proactive partitioning.

1. INTRODUCTION

Dynamically adaptive simulations based on structured adaptive mesh refinement (SAMR) techniques can yield highly advantageous ratios for cost/accuracy when compared to methods based

[⋆] The research presented in this paper is supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA-0120934, CNS-0305495, by DOE ASCI/ASAP via grant numbers PC295251 and 82-1052856, and by the DoE SciDAC program via grant number DE-FC02-01ER41184.

on static uniform approximations. Parallel/distributed SAMR implementations lead to interesting research problems in dynamic resource allocation, data-distribution, load balancing, communication, and coordination. Furthermore, the underlying Grid infrastructure is dynamic and heterogeneous in nature. As a result, configuring, managing, and optimizing the execution of dynamic SAMR applications to exploit the computational power of the heterogeneous Grid environment remains a significant challenge.

GridARM [(1)] is an autonomic runtime management framework that monitors application and system state and provides adaptation strategies to optimize the performance of SAMR applications in distributed and dynamic Grid execution environments. Key GridARM components presented in this paper include reactive and proactive partitioning strategies based on application/system runtime state and performance estimation. The experimental evaluation of these runtime management schemes for different SAMR kernels and system workloads demonstrates an improvement in overall application performance.

[Fig. 1 about here.]

The conceptual model of the GridARM framework is illustrated in Fig. 1. Application sensors monitor the state of the SAMR grid hierarchy and the nature of its refined regions, and characterize the application in terms of metrics such as computation/communication requirements, storage requirements, activity dynamics, and the nature of adaptations. System sensors characterize the current state of the underlying computational resources in terms of CPU, memory, bandwidth, availability, and access capabilities. The characterization of current application/system state drives predictive performance functions and models that can estimate application performance. The GridARM deduction engine uses current state and adaptation policies to formulate prescriptions for algorithms, configurations, and parameters, and defines normalized work and resource metrics that characterize the runtime state. Using these metrics as inputs, the autonomic runtime manager then defines a hierarchical distribution mechanism, configures and deploys appropriate partitioners at each level of the hierarchy, schedules and maps application working-sets onto virtual resources, and tunes the SAMR application within the Grid environment.

The rest of this paper is organized as follows. Section 2 describes the design and evaluation of the application-sensitive partitioning framework. Section 3 discusses the system-sensitive partitioning strategy. Section 4 details the implementation and evaluation of the SAMR proactive partitioning schemes. Section 5 presents concluding remarks.

2. AUTONOMIC APPLICATION-SENSITIVE PARTITIONING

The choice of an appropriate partitioner depends on the application configuration and state, since partitioners typically optimize a subset of the application requirements at the expense of others. SAMR applications are dynamic in nature (i.e., their requirements change with time as the application proceeds) and hence the current SAMR application state can only be determined at runtime. ARMaDA (2) dynamically selects and configures partitioning algorithms at runtime to optimize the overall performance of SAMR applications. The partitioners used include a selection from software tools such as GrACE (Grid Adaptive Computational Engine) (3) and Vampire (4).

2.1. Application State Characterization and Partitioner Selection

ARMaDA builds on our previous work (5) that experimentally studied the behavior of structured domain-based partitioners and defined an approach for characterizing the state of SAMR applications. In ARMaDA, the current application state is monitored using simple geometric operations from the structure of the SAMR grid hierarchy, expressed as sets of bounding boxes at various levels of refinement. Three primary metrics are used: (i) computation-to-communication ratio (“CCratio”) determines whether the application state is computationally-intensive or communication-dominated; (ii) application dynamics (“Dynamics”) estimate the rate of change of application refinement patterns; and (iii) nature of adaptations (“Adapt”) captures the adaptation pattern, i.e., whether refinements are scattered or localized. These are computed as follows.

$$\text{CCratio} = \frac{\sum \text{(Volume of bounding boxes)}}{\sum \text{(Surface area of bounding boxes)}} \tag{1}$$

$$\text{Dynamics} = \text{Size of } (\text{Current state boxes} \cap \text{Previous state boxes}) \tag{2}$$

$$\text{Adapt} = \frac{\text{Volume of refinement regions}}{\text{Domain volume}} * \text{Number of refinement patches} \tag{3}$$

The ARMaDA framework maintains a history of application state by storing the structure of the SAMR grid hierarchy for two preceding regrid steps. This avoids possible thrashing due to very frequent state changes. If M_r is a metric computed at regrid step r, its normalized metric ratio is computed using this 3-step sliding window as

$$\text{Mratio} = \frac{\text{currM}_r * \text{currM}_{r-2}}{(\text{currM}_{r-1})^2} \tag{4}$$

Three normalized ratios are computed corresponding to the three metrics, viz., computation/-communication ratio “Cratio”, application dynamics ratio “Dratio”, and adaptation ratio “Aratio”. Using low and high thresholds and application-dependent weights, these ratios are combined to characterize the state of the SAMR application. The characterized application state is then mapped to appropriate partitioners using policies derived heuristically from previous research [(5)].

The ARMaDA framework configures the selected partitioner with appropriate partitioning parameters (such as partitioning granularity) and invokes it to partition and load balance the SAMR grid hierarchy. The granularity is based on the requirements of the current application state, though it may be overidden by a user defined value. ARMaDA uses efficient and inexpensive mechanisms and provides optimizations to ensure that the runtime overheads of state characterization and partitioner selection do not offset the benefits of adaptation.

2.2. Evaluation of Application-Sensitive Partitioning

The experimental evaluation of ARMaDA consists of measuring the overall execution times for the different partitioners used individually, including SFC, G-MISP+SP, pBD-ISP, and their adaptive combinations. Only the partitioning strategy and associated granularity are varied. All other parameters are kept constant.

The first experiment is conducted on 32 processors of "Frea", a 64-node Linux Beowulf cluster at Rutgers University, using the VectorWave2D[3] application. The application uses a base grid of size 128*128 with 3 levels of factor 2 space-time refinements. Regriding is performed every 4 time-steps at each level and the application runs for 60 coarse level time steps. The VectorWave2D application is primarily computation-dominated, requiring good load balance and reduced communication and data migration costs. SFC and pBD-ISP partitioners optimize communication and data migration, while G-MISP+SP gives good load balance. As shown in Table I, the ARMaDA

[3] The VectorWave2D application forms a part of the Cactus 2-D numerical relativity toolkit solving Einstein's and gravitational equations.

partitioner with SFC start improves performance by 26.19% over the slowest partitioner, and the overhead is 0.0616% of the total time, which is negligible.

[Table I about here.]

The second experiment is conducted on 64 processors of “Blue Horizon”, the NPACI IBM SP2 system at the San Diego Supercomputing Center, using the RM2D[4] application. RM2D uses a 128*32 base grid and executes for 60 iterations with 3 levels of factor 2 refinements and regriding every 4 time-steps at each level. The execution speedup provided by ARMaDA (Table I) is 4.66%, 11.32%, and 27.88% over pBD-ISP, G-MISP+SP, and SFC partitioners respectively. The overhead is 0.415 seconds and is minimal compared to overall execution times.

3. REACTIVE SYSTEM-SENSITIVE PARTITIONING

3.1. Characterizing System State

The GridARM runtime management framework reacts to system capabilities and current system state to select and tune distribution parameters while dynamically partitioning and load balancing the SAMR grid hierarchy. Current system state is obtained at runtime using the NWS (6) resource monitoring tool. If the total work W is to be distributed among K processors, the work W_i assigned to the ith processor can be computed as $W_i = CR_i \times W$, where CR_i represents the combined work partitioning ratio for processor i computed using current system information such that

$$CR_i = w_c R_i^C + w_m R_i^M + w_b R_i^B \quad \text{and} \quad \sum_{i=0}^{K} CR_i = 1 \tag{5}$$

R_i^C, R_i^M, and R_i^B are work partitioning ratios based on CPU load, available memory, and link bandwidth, respectively. w_c, w_m, and w_b are the application-specific weights associated with relative

[4] RM2D is the 2-D compressible turbulence kernel solving the Richtmyer-Meshkov instability.

CPU, memory, and bandwidth availabilities such that $w_c + w_m + w_b = 1$. Note that

$$\sum_{i=0}^{K} R_i^C = \sum_{i=0}^{K} R_i^M = \sum_{i=0}^{K} R_i^B = 1 \tag{6}$$

[Fig. 2 about here.]

3.2. Evaluation of Reactive System-Sensitive Partitioning

The system-sensitive partitioner (7) is evaluated on a 32-node Linux-based workstation cluster using the RM3D[5] application. The application uses 3 levels of factor 2 space-time refinements on a base mesh of size 128*32*32. The experimental setup consists of a synthetic load generator (for simulating heterogeneous loads on the cluster nodes) and an external resource monitoring system (i.e., NWS). The evaluation compares the execution time and load balance generated for the system sensitive partitioner with those for the GrACE (3) infrastructure which distributes the workload equally among processors. As illustrated in Fig. 2, system-sensitive partitioning reduces the execution time by about 18% for 32 nodes. Figure 3 shows the system-sensitive workload assignment on a cluster consisting of 4 nodes with relative capacities 16%, 19%, 31%, and 34%. The system-sensitive partitioner reduces imbalances by about 45%.

4. PROACTIVE SAMR PARTITIONING STRATEGIES

Parallel/distributed SAMR applications are sensitive to CPU, memory, and bandwidth requirements, and their performance may degrade severely due to increased CPU and/or network loads and reduced available memory. There are three extreme cases in Eq. (5) which lead to three different proactive partitioning strategies, developed for the RM3D application: (i) *CPU-based runtime*

[5] RM3D is the 3-D Richtmyer-Meshkov instability solver encountered in compressible fluid dynamics and has been developed by Ravi Samtaney as a part of virtual test facility at Caltech ASCI/ASAP Center.

partitioning: $w_c = 1, w_m = w_b = 0$, $CR_i = R_i^C$ and work is partitioned based on CPU load status; (ii) *Memory-based runtime partitioning:* $w_m = 1$, $w_c = w_b = 0$, $CR_i = R_i^M$ and work is partitioned based on available memory; and (iii) *Bandwidth-based runtime partitioning:* $w_b = 1$, $w_c = w_m = 0$, $CR_i = R_i^B$ and work is partitioned based on link bandwidth.

4.1. CPU-based Runtime Partitioning

Performance Functions (PF) (8) describe the behavior of system or application in terms of changes in one or more of its attributes. The execution time of a computation-intensive program increases linearly with the number of jobs sharing the same processor. The SAMR application execution time can be estimated as a function of CPU load, application work W, and refinement level LV, given by $T_i = T \times L_i = PF_C(W_i, LV_i) \times L_i$, where L_i represents the length of the CPU queue for processor i. The RM3D time performance function is empirically defined in Eq. (7), where a_i is a heuristic coefficient derived from previous research (9).

$$\begin{aligned} T &= PF_C(W, LV) \\ &= a_0 + a_1 W + a_2 LV + a_3 W \times LV + a_4 W^2 + a_5 LV^2 \\ &\quad + a_6 W^2 \times LV + a_7 W \times LV^2 + a_8 W^2 \times LV^2 \end{aligned} \tag{7}$$

To minimize disparity in execution times, the work partitioning ratio is adjusted at runtime such that the execution time at the next time step is identical within an acceptable tolerance. The adjustment factor for each processor is defined by

$$f_i(t) = \frac{T_{avg}(t)}{T_i(t)} \quad \text{where} \quad T_{avg}(t) = \frac{\sum_{i=0}^{K} T_i(t)}{K} \tag{8}$$

where $T_i(t)$ and $T_{avg}(t)$ are the estimated execution time for processor *i* and the average estimated execution time, respectively, at time step t. Once the adjustment factor is determined, the work partitioning ratio for processor *i* at the next time step is computed and normalized as

$$R_i^C(t+1) = R_i^C(t) \times f_i(t) \quad \text{and} \quad R_i^C(t+1)' = \frac{R_i^C(t+1)}{\sum_{i=0}^{K} R_i^C(t+1)} \tag{9}$$

4.2. Memory-based Runtime Partitioning

The performance of SAMR applications may degrade on heavily loaded processors that have little available memory because of frequent page faults. The memory-based partitioning strategy optimizes performance by minimizing the number of page faults and balancing work among processors. The memory function for RM3D is empirically defined as $AM = PF_M(W) = a_0 + a_1 W$, where a_0 = 8187.5036, a_1 = 0.1348959, and AM is the memory usage corresponding to work W.

The processors are divided into different groups according to their available physical memory space. Let M_i be the available physical memory on processor *i* and MT_1 and MT_2 denote a two-level threshold that describes the memory characteristics. If $M_i < MT_1$, processor *i* belongs to the low memory group (X^-) with frequent page faults. If $M_i > MT_2$, processor *i* belongs to the high memory group (X^+) with rare or normal page faults. If M_i is between MT_1 and MT_2, processor *i* is in moderately loaded group (X) with occasional page faults. The number of processors in group X^-, X^+, and X are represented by N^-, N^+, and N, respectively. When both N^+ and N^- are greater than zero, work assigned to processors in group X^- is partially transferred to processors in group X^+, with no change in work for processors in group X. In other cases, the current work assignment is kept unchanged. Let W^- be the entire work assigned to the processors in group X^- and let P_1

be the transferring percentage. Then, $W^- \times P_1$ denotes the work to be transferred while avoiding overloading the processors in group X^+. To ensure an even transfer of work, let U^- denote the unit of work being transferred to one processor in group X^+ such that

$$U^- = \frac{W^- \times P_1}{N^+} \quad \text{where} \quad W^- = \sum_{i=0}^{N^-} W_i \tag{10}$$

In order to achieve the desired optimization, a processor in group X^+ must guarantee that its available memory space is greater than MT_1 after it accepts the work transfer. In doing so, it must estimate the memory usage of its current work W_i and additional work after the transfer. Let the work for processor *i* after the transfer be W_i' such that $W_i' = W_i + U^-$. In order to avoid overloading after the transfer, each processor in group X^+ must satisfy the following condition.

$$[M_i - \{PF_M(W_i') - PF_M(W_i)\}] > MT_1 \tag{11}$$

The processors in group X^+ are sorted in the ascending order of available memory M_i. Upon substituting $W_i' = W_i + U^-$ into Eq. (11), the checks are performed for processor *i* (with least available memory onwards) in group X^+ to test if the condition in Eq. (11) can be met. If the condition cannot be met, further reduction to the work transfer (U^-) is performed and the remaining work is left for the next processor. If the condition can be met, the algorithm attempts to transfer the remaining work from the previous processor as well. A similar check as in Eq. (11) is performed for the remaining work and further reduction may be possible to avoid overloading processors in group X^+. After estimating the work transfer, each processor obtains new work W_i' and the memory-based work partitioning ratios for the next time step are computed as

$$R_i^M = \frac{W_i'}{\sum_{i=0}^{K} W_i'} \tag{12}$$

4.3. Evaluation of CPU-based Proactive Partitioning

The proactive runtime partitioning strategies are integrated within the GridARM framework and are evaluated using the RM3D application on Beowulf clusters at Rutgers University and University of Arizona. Three scenarios are established for this evaluation: (i) *Lightly loaded scenario*: 75% of the processors are lightly loaded and the other 25% are heavily loaded; (ii) *Moderately loaded scenario*: 50% are lightly loaded and the other 50% are heavily loaded; and (iii) *Heavily loaded scenario*: 25% are lightly loaded and the other 75% are heavily loaded.

[Table II about here.]

In this evaluation, a synthetic program adapts the CPU load dynamics among processors in order to establish the three different load scenarios. Table II presents the performance gain for the RM3D application on 32 and 64 processors using the CPU-based strategy under different load scenarios. Without CPU load adaptation, the work is assigned almost evenly among processors regardless of their CPU load status that leads to longer application execution time when some processors are heavily loaded. However, when the CPU-based proactive partitioning algorithm is used, work is assigned to processors based on their CPU load status and the performance of RM3D improves significantly. Moreover, better performance can be achieved under lightly and moderately loaded scenarios while there are not enough lightly loaded processors to accept more work for the heavily loaded scenario. Furthermore, better performance can be obtained with large number of processors since the work assigned to each processor is reduced.

4.4. Evaluation of Memory-based Proactive Partitioning

In this evaluation, the memory availability of processors is controlled by a synthetic memory consuming program in order to establish the three different memory usage scenarios. Figure 4 illustrates the memory availability on 8 processors for the moderately loaded scenario and the corresponding work assignment using the memory-based proactive partitioning approach. Table III presents the performance gain on 8 processors using the memory-based proactive partitioning strategy. Without this scheme, the work is partitioned evenly among processors regardless of their memory availability and thus the RM3D application experiences long delays. However, with the memory-based algorithm, the application work is reassigned to processors according to memory availability, resulting in a significant reduction in execution time. The results also demonstrate that better performance is achieved under moderately and heavily loaded scenarios since page faults occur frequently due to lesser available memory.

[Fig. 3 about here.]

[Table III about here.]

5. CONCLUSION

This paper presented application and system sensitive reactive and proactive partitioning strategies to optimize the performance of SAMR applications in distributed and dynamic Grid execution environments. These strategies form a part of the GridARM autonomic runtime management framework. The current application state is characterized in terms of application-level metrics such as computation/communication requirements, storage requirements, activity dynamics, and the nature of adaptations. The performance prediction functions are experimentally formulated in terms of the current system state such as CPU load, available memory, and bandwidth. The GridARM framework uses the runtime state information to redistribute and load-balance the application in order to address changing application/system requirements and maximize performance. The experimental evaluation of these runtime management strategies for different SAMR kernels and system workloads demonstrates an improvement in overall application performance.

REFERENCES

1. S. Chandra, M. Parashar, and S. Hariri. "GridARM: An Autonomic Runtime Management Framework for SAMR Applications in Grid Environments", Autonomic Applications Workshop, *High Performance Computing* (HiPC'03), India, pp. 286–295, December 2003.

2. S. Chandra and M. Parashar. "ARMaDA: An Adaptive Application-Sensitive Partitioning Framework for Structured Adaptive Mesh Refinement Applications", *Proc. of Parallel and Distributed Computing Systems* (PDCS'02), Cambridge, MA, pp. 446–451, November 2002.

3. M. Parashar, http://www.caip.rutgers.edu/TASSL/Projects/GrACE, GrACE homepage.

4. J. Steensland, http://www.caip.rutgers.edu/˜johans/vampire, Vampire homepage.

5. J. Steensland, S. Chandra, and M. Parashar, "An Application-Centric Characterization of Domain-Based SFC Partitioners for Parallel SAMR", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 12, pp. 1275–1289, December 2002.

6. R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Future Generation Computing Systems*, 1999.

7. S. Sinha and M. Parashar, "Adaptive System-Sensitive Partitioning of AMR Applications on Heterogeneous Clusters", *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, Kluwer Academic Publishers, Vol. 5, Issue 4, pp. 343–352, 2002.

8. H. Zhu, M. Parashar, J. Yang, Y. Zhang, S. Rao, and S. Hariri. "Self Adapting, Self Optimizing Runtime Management of Grid Applications using PRAGMA", *Proc. of NSF NGS Program Workshop, IEEE/ACM 17th IPDPS*, Nice, France, CDROM, 7 pages, April 2003.

9. S. Chandra, S. Sinha, M. Parashar, Y. Zhang, J. Yang, and S. Hariri. "Adaptive Runtime Management of SAMR Applications", *Proc. of High Performance Computing* (HiPC'02), LNCS, India, Vol. 2552, pp. 564–574, December 2002.

LIST OF TABLES

Table I

VectorWave2D on Frea		RM2D on Blue Horizon	
32 processors		64 processors	
Partitioner evaluated	Execution time (sec)	Partitioner evaluated	Execution time (sec)
SFC	637.478	SFC	264.041
G-MISP+SP	611.749	G-MISP+SP	214.745
pBD-ISP	592.05	pBD-ISP	199.738
ARMaDA with SFC start	470.531	ARMaDA with G-MISP+SP start	190.431

Table II

Scenarios	No. of procs.	Execution time without CPU adaptation (sec)	Execution time with CPU adaptation (sec)	Percentage Improvement
Lightly loaded	32	4908.79	2901.1	40.9%
	64	4904.78	2827.29	42.36%
Moderately loaded	32	4976.78	3378.65	31.35%
	64	5049.72	3281.31	35.02%
Heavily loaded	32	5170.52	4140.56	20.45%
	64	5119.89	3587.89	29.92%

Table III

Scenarios	Execution time without memory adaptation (sec)	Execution time with memory adaptation (sec)	Percentage Improvement
Lightly loaded	6922.14	5210.87	24.72%
Moderately loaded	15890.47	7401.61	53.42%
Heavily loaded	16962.1	8284.84	51.16%

LIST OF FIGURES

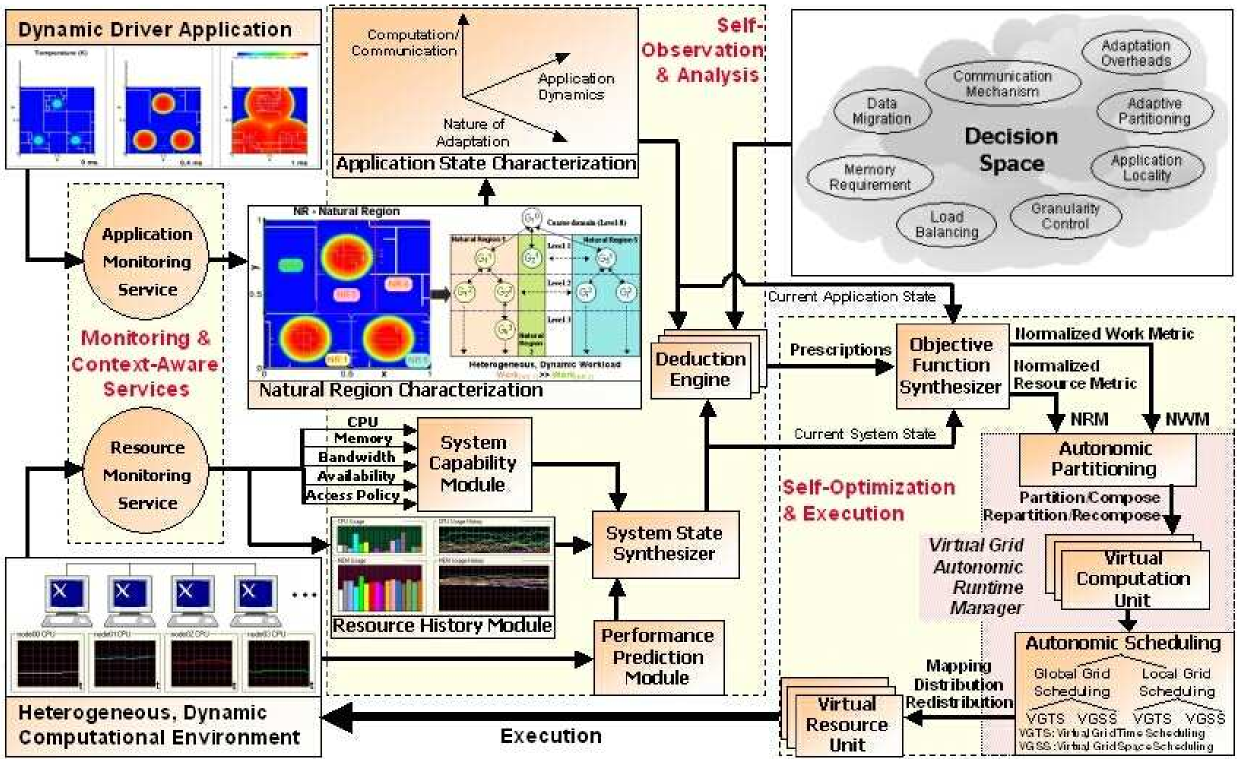
Dynamic Driver Application
Computation/Communication
Application Dynamics
Nature of Adaptation
Application State Characterization
Self-Observation & Analysis
Decision Space
Adaptation Overheads
Communication Mechanism
Data Migration
Adaptive Partitioning
Memory Requirement
Application Locality
Load Balancing
Granularity Control
Application Monitoring Service
NR - Natural Region
Heterogeneous, Dynamic Workload
Natural Region Characterization
Monitoring & Context-Aware Services
Current Application State
Deduction Engine
Prescriptions
Objective Function Synthesizer
Normalized Work Metric
Normalized Resource Metric
NRM
NWM
Current System State
Resource Monitoring Service
CPU
Memory
Bandwidth
Availability
Access Policy
System Capability Module
System State Synthesizer
Self-Optimization & Execution
Autonomic Partitioning
Partition/Compose
Repartition/Recompose
Virtual Grid Autonomic Runtime Manager
Virtual Computation Unit
Resource History Module
Performance Prediction Module
Autonomic Scheduling
Global Grid Scheduling
Local Grid Scheduling
VGTS
VGSS
VGTS: Virtual Grid Time Scheduling
VGSS: Virtual Grid Space Scheduling
Mapping
Distribution
Redistribution
Virtual Resource Unit
Heterogeneous, Dynamic Computational Environment
Execution

Fig. 1

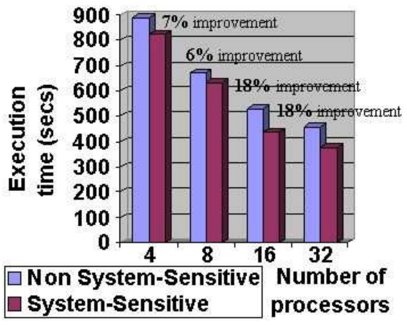
Execution time (secs)
900
800
700
600
500
400
300
200
100
0
7% improvement
6% improvement
18% improvement
18% improvement
4
8
16
32
Number of processors
Non System-Sensitive
System-Sensitive

Fig. 2

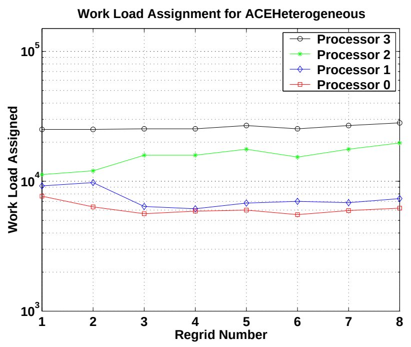
Work Load Assignment for ACEHeterogeneous
Processor 3
Processor 2
Processor 1
Processor 0
Work Load Assigned
10^5
10^4
10^3
1
2
3
4
5
6
7
8
Regrid Number

Fig. 3

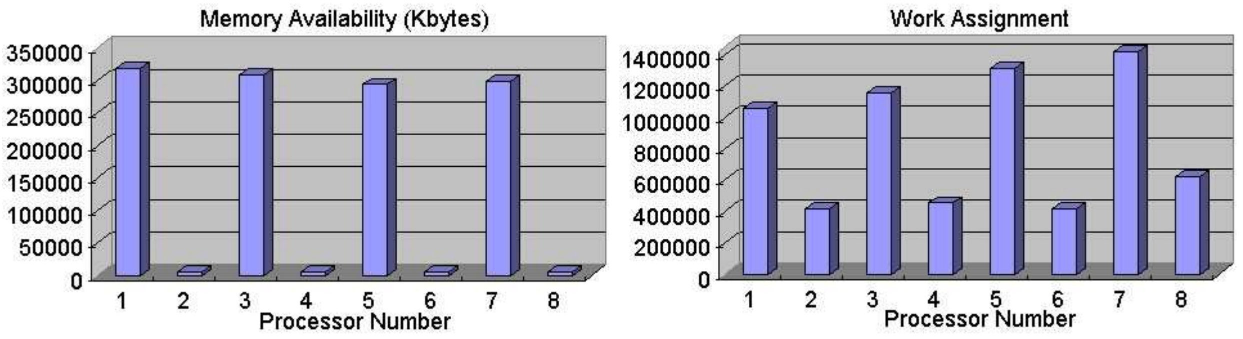
Memory Availability (Kbytes)
350000
300000
250000
200000
150000
100000
50000
0
1
2
3
4
5
6
7
8
Processor Number
Work Assignment
1400000
1200000
1000000
800000
600000
400000
200000
0
1
2
3
4
5
6
7
8
Processor Number

Fig. 4