Pragma: An Infrastructure for Runtime Management of Grid Applications*

Manish Parashar
The Applied Software System Laboratory
Rutgers, The State University of New Jersey
Piscataway, NJ 08854, USA
parashar@caip.rutgers.edu

Salim Hariri
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, USA
hariri@ece.arizona.edu

Abstract

This paper presents an overview of Pragma, an adaptive runtime infrastructure 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 overarching motivation for this research is to enable the next generation of very large-scale dynamically adaptive scientific and engineering simulations on widely distributed and highly heterogeneous and dynamic execution environments such as the computational "grid". Pragma combines 4 key components: system characterization and abstraction component, application characterization component, active network control, and policy-base. The design of Pragma is driven by three astrophysical simulations chosen to be representative of a wide variety of important simulations and to expose many of the problems presently encountered (and currently unsolved) by computational physicists. The design, prototype implementation, and preliminary evaluations of Pragma components are presented.

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" which will integrate scalable distributed (and heterogeneous) computing with interactive control and computational steering, collaborative analysis, visualization, and scientific databases and data archives. These simulations will provide new and important 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 the Pragma infrastructure is to realize a next-generation adaptive runtime infrastructure 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 Pragma project addresses 3 key research challenges:

1. Formulation of predictive performance functions that hierarchically combine analytical, experimental and empirical performance models for individual elements of a heterogeneous, distributed computational environment, and use these functions along with current system/network state information to anticipate the operations and expected performance of applications for a given workload and system configuration.

2. Development of mechanisms for monitoring and characterizing the state of adaptive applications and abstracting their current computational, communication and storage requirements, and using this information to maximize application efficiency and performance.

3. Design, development and deployment of an active control network combining application sensors and actuators and application management agents capable of configuring application and execution environment at runtime, allocating and setting up required resources, monitoring application and system state, proactively and reactively adapting the application and execution environment to satisfy application requirements, maintaining application quality of service, improving performance and/or respond to system failures.

The design, development and evaluation of the Pragma framework is being conducted in the context of a real-

*The work presented in this paper is supported by the National Science Foundation NGS program via grant EIA-0103674

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

world astrophysical hydrodynamics simulation using adaptive mesh refinement. Three-dimensional hydrodynamic simulations are the key to understanding a wide variety of astrophysical problems from the formation of structures in the universe at the largest scales to the formation of stars and their eventual explosion as supernovae. Accurate solutions of realistic models offer the potential of dramatic insights into complex astrophysical phenomena. This paper present a design overview, and prototype implementation and preliminary evaluations of Pragma components. The rest of this paper is organized as follows. Section 2 outlines the problem description enabling realistic simulations of astrophysical phenomenon. Section 3 describes the different components of the Pragma framework for proactive and reactive run-time management of grid applications. A Pragma prototype case study on an adaptive meta-partitioner for AMR applications is presented in Section 4. Section 5 presents concluding remarks.

2 Problem Description: Enabling Realistic Simulations of Astrophysical Phenomenon

The design of Pragma is driven by specific problems in astrophysical simulations. The problems are chosen to be representative of a wide variety of important simulations and to expose many of the problems presently encountered (and currently unsolved) by computational physicists. Numerical simulation of galaxy formation is one of the most demanding in computational astrophysics. Galaxies are believed to have formed hierarchically; objects of progressively larger mass merge and collapse to form new systems. Such a hierarchical build-up lacks any simplifying symmetries and a full three-dimensional simulation is required. Similarly, multidimensional hydrodynamics in supernovae from massive stars involve highly asymmetrical and aspherical explosions and debris fields, and also require 3-D simulations with a wide range of spatial scales.

A key challenge in the astrophysical simulations outlined above is that the physics represented in such calculations 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 (MIT), which allows the grid resolution to adapt to a local estimate of the error in the solution. With AMR, the number of zones and their location in the problem space is continuously changing. With MIT, the frequency with which the data in the zones is updated can vary widely. Furthermore, a solution of the system's self-gravity requires the combination of hyperbolic hydrodynamic equations and elliptic Poisson's equation with different 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 astrophysical simulations thus lead to interesting computational and computer science challenges in dynamic resource allocation, data-distribution and load balancing, communications and coordination, and resource management. As a result, key requirements for the Pragma 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 run-time management a significant challenge.

3 Pragma: A Framework of Proactive & Reactive Runtime Management of Grid Applications

Pragma is composed of 4 key components: system characterization and abstraction component, application characterization component, active network control, and policy-base. The system characterization and abstraction component is responsible for abstracting the current state of the underlying computational environments and for performing a predictive analysis of its behavior. The application characterization components abstract the current state of the adaptive (AMR) application in terms of its communication and computational requirements and the natures and dynamics of its grids. The active network control composed sensors, actuators and management/policy agents of adaptive runtime control. Finally, the policy-base is a programmable database of adaptation polices that be used by the agents and will drive the overall adaptation process.

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

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

satisfied and its performance optimized. 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. The Pragma system characterization component builds on existing infrastructure, such as NWS [9].

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 application [5].

We illustrate our approach by using it to model and analyze a simple networked system. Our goal is to generate the PF that describes the overall behavior of the system with respect to the desired metric. This example system consists of two Computers (PC1 and PC2) that are connected through an Ethernet switch. We assume that PC1 performs a matrix multiplication and upon completion, PC1 sends the result to PC2 through the Switch. PC2 performs the same matrix multiplication function and returns the result back to PC1 where the process is repeated. Using our approach we want to find the performance function to analyze the response time (delay) for the whole application. For simplicity, we only consider the data size attribute and determine the application response time with respect this attribute. For each component, we measure the task processing time in terms of data size, and then feed these measurements to a neural network to obtain the corresponding PF. For our example, we obtain the following three PFs; PF_1 (i=1) denotes the PF associated with PC1, PF_2 (i=2) denotes PC2, and PF_3 (i=3) denotes the switch:

$$PF_i = \sum_{j=1}^{3} \left(\frac{a_j}{1 + \exp^{-(b_j + c_j D)}} \right) + d_i \qquad (1)$$

where, a_j, b_j, c_j, d_i are constants and D is the data size. The end-to-end performance function of whole system ($PF_{overall}$) is the summation of all the PFs:

$$PF_{OVERALL} = PF_{PC1} + PF_{SWITCH} + PF_{PC2} \qquad (2)$$

Data Size (bytes)	$PF_{End-to-End}$	Measured end-to-end Delay	%Error
200	8.2759e-04	8.3187e-04	0.515
400	0.0011815	0.0011288	4.67
600	0.0014516	0.0015312	5.2
800	0.0017969	0.0018809	4.46
1000	0.0021705	0.00223055	2.7

Table 1. Accuracy of the Performance Functions.

Table 1 shows the error incurred in modeling the end-to-end delay based on PF modeling approach is roughly between 0.5 - 5%. Details about the PF-based approach for modeling large-scale distributed systems are found in [3].

3.3 Application Characterization

Dynamically adaptive application, such as the AMR-based astrophysical simulation targeted in this proposal, adapt their behavior based on the current state of the physical phenomenon being simulated. Unlike static applications where requirements are typically known a priori, the requirements of these applications change as the application progresses and therefore can only be determined at runtime. The objective of Pragma's application characterization module is to abstract the state of the AMR application in order to determine its current computational, communication, and storage requirements. This information can then be used to determine an appropriate decomposition of the application and mapping of the computations to available processing elements of the computational environment, and to drive the selection of appropriate algorithms and implementations both at the application level (solvers, precondi-

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

tioners) as well as the system level (communication mechanism). Note that the application characterization is architecture/system independent.

3.4 Agent-based Runtime Adaptation

The underlying mechanisms for adaptive run-time management of grid applications is 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.

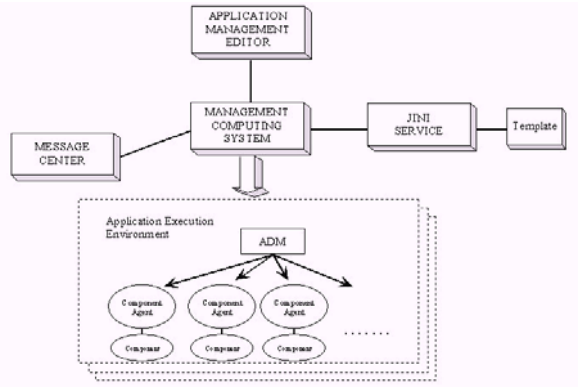

Figure 1. CATALINA Architecture.

3.4.1 Agent-based Application Management

The Pragma application management approach builds on and extends the management agents provided by CATALINA [2]. CATALINA provides application developers with all the tools required to specify the appropriate control and management schemes to maintain any quality of service requirement or application attribute/functionality (e.g., performance). It also provides core management services to enable efficient proactive management of a wide range of network applications.

The architecture of CATALINA is shown in Figure 1. The Application Management Editor (AME) tool provides application developers with the services required for specifying and characterizing application requirements in terms of performance, fault-tolerance and security, and for specifying the appropriate management scheme for maintaining application requirements. The next step utilizes the management services provided by the Management Computing System (MCS) to build the appropriate application execution environment that can dynamically control the allocated resources to maintain application requirements during its execution. The MCS assigns an Application Delegated Manager (ADM) to manage one or more application attributes (performance, fault, security, etc.). For each task/component in the application, the ADM launches an appropriate Component Agent (CA) to monitor execution using appropriate component sensors. The CA intervenes whenever component execution on the assigned machine cannot meet its requirements using component actuators that can suspend, save component execution state, or migrate the component execution to another machine. For example, to manage the component performance, ADM may use active redundancy, passive redundancy, or may migrate the task to a faster machine. The appropriate management scheme is selected at runtime.

To configure the application execution environment, the MCS searches for an appropriate template in the template database that can meet all application requirements. The template can be viewed as a blueprint of the application execution environment. The CATALINA template registry is being updated to use a JINI-based open architecture to allow third party template registration and discovery. CATALINA uses a Message Center (MC) for all the communications between its modules and agents. In the MC, every component is assigned a port which acts as its mailbox. Every message directed to a component is placed on this mailbox.

3.4.2 Sensors and Actuators for Active Adaptation

In addition to the management agents, sensors and actuators need to be embedded within the application and/or system software, as they define the adaptation interface and implement the mechanics of adaptation. Application level sensors and actuators are embedded within the application source using high level programming abstractions. Furthermore, the 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 [4] and others [8]. System level sensors build on existing instrumentation-based systems such as NWS, ReMoS, and Autopilot.

3.5 Adaptation "Policy" Knowledge-Base

The adaptation policy base maintains polices used by the management agents to drive decision-making during runtime application management and adaptation. Policies encode rules, heuristics and experiences that relate system and application state abstraction to system/application configurations, algorithms and mechanisms. For example: *"If on a networked cluster and AMR application is in octant VI use latency-tolerant communication"*, *"If machine architecture is X and application is in octant Y use data-decomposition*

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

Z" or *"If cache size of Y use refined grid components no larger than Q"*. This machinery enables us to construct a library of selection strategies (templates) that encode, for example, how to partition and map the grid hierarchy for a strictly flux-conservative application with scattered highly dynamic refinements patterns on a clusters of SMP's. Programmability of the knowledge base will allow rules to be modified, adapted and extended. On the management agent side, the policy knowledge base will present an associative interface that allows the agents to formulate partial queries and use fuzzy reasoning. The policy knowledge-base construction leverages off existing case-based reasoning systems and inference engines.

3.6 Adaptive Application Management

The key goal of Pragma is to develop policies and mechanisms for both "system sensitive" and "application sensitive" runtime adaptations of AMR applications. The former is driven by current system state and system performance predictions while the latter is based on current application state. The management agents in the active control network then use these policies to actively manage the application.

System sensitive application management uses current and predicted system state characterization to make application adaptation decisions. For example, the information about the current load and available memory may determine the granularity of the mapping of the application components to processing nodes, while available communication bandwidths will determine the type of 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. Application sensitive adaptations 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 configurations of partitioning and load-balancing algorithms, communication mechanisms, etc.

4 Case Study: Engineering an Adaptive Meta-Partitioner for AMR Applications

This section presents the design and operation of a prototype Pragma component that implements an adaptive system and application sensitive meta-partitioner for AMR applications. The overall motivation of the adaptive meta-partitioner is to dynamically select the most appropriate partitioning strategy at runtime, based on current application and system state. Our preliminary studies with such an adaptive partitioner have shown that using runtime adaptive application management can significantly decrease application execution time. Many factors contribute to the overall runtime of an AMR application. Consequently, optimizing the runtime of these applications requires identifying an optimal set of these parameters. Furthermore, the relationships between the contributing factors might be highly intricate, and depend on current application and system state.

4.1 Defining Partitioning Quality

The partitioning requirements for an adaptive application (and the performance of a particular partitioner) depend on the current state of the application and the computing environment. Therefore, it is of little consequence to discuss the absolute "goodness" of a certain partitioning technique. We base our characterization of partitioning behavior on the tuple $\{$*partitioner, application, computer system*$\}$, (PAC). Furthermore, we define a five-component metric to evaluate each PAC. The goal of the metric is to capture the overall runtime behavior of the partitioner. We believe that this is critical to understanding the suitability of a particular partitioner or *why* one PAC works better than another PAC. The proposed metric for characterizing the quality of a PAC for the adaptive SAMR meta-partitioner include *Communication requirements*, *Load imbalance*, *Amount of data migration*, *Partitioning time*, and *Partitioning induced overheads*. Optimizing all these components implies conflicting objectives. For example, optimizing the first two components together constitutes an NP-hard problem. Partitioners typically optimize a subset of the components at the expense of others. Our goal in defining this metric is to be able to readily determine the trade-offs for each partitioning technique.

4.2 Defining Application State: The Octant Approach

Our characterization of AMR applications is based on the Octant Approach [7] (Figure 2). In this approach, the AMR application state is classified based on (a) its current adaptation pattern (scattered or localized), (b) the current dynamics of the phenomenon (e.g. a moving shock giving rise to rapidly changing adaptations), and (c) whether its runtime is dominated by computations or communications. Application adaptation patterns determine the nature of the computational domain, the granularity of computations, and the required interactions between application components. The application dynamics determine the rate (and possibly nature) of changes in computation/communication requirements and how often load will have to be balanced. Finally, the third dimension is a classical characterization of the application as being dominated by computations or communications. Applications may start in one octant, then, as

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

solution progresses, migrate to others.

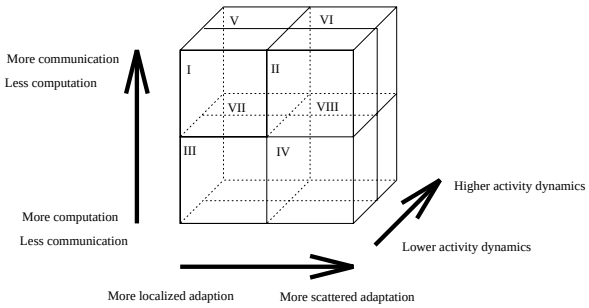

Figure 2. The *octant approach* for characterizing application state based on its adaptation pattern, communication/computation requirements and activity dynamics.

4.3 Adaptive Selection of Partitioning Techniques

Previous research has investigated the selection of the appropriate partitioner for a certain combination of the application and computer system to reduce runtime. However, the PAC tuple is a dynamic entity. The (A)pplication and the (C)omputer system components are obviously changing. The *choice* of the (P)artitioner as static can be a serious limitation. Consequently, we define the PAC relationship as $P_t = f(A_t, C_t)$, indicating that the partitioning technique P selected at a given time t should be a function of the state of the application A and the computer system C at that time.

The adaptive meta-partitioner for SAMR applications is based on this idea. The runtime environment is characterized using the octant approach and current application and system state. Based on the octant state, the most appropriate partitioning technique is selected from a database of available partitioning techniques, configured with appropriate parameters such as partitioning granularity and threshold, and then invoked to partition the SAMR grid hierarchy.

4.4 Application Sensitive Adaptations

Application state information is used to identify the partitioning requirements and to select the most appropriate partitioner from a suite of available patch and domain based partitioners. Available partitioners include *Space-Filling Curve based Partitioner (SFC), Variable Grain Geometric Multilevel Inverse Space-Filling Curve Partitioner (G-MISP), Variable Grain Geometric Multilevel Inverse Space-Filling Curve Partitioner with Sequence Partitioning (G-MISP+SP), p-Way Binary Dissection Inverse Space-Filling Curve Partitioner (pBD-ISP),* and *Pure Sequence Partitioner with Inverse Space-Filling Curve (SP-ISP).* An application-centric characterization of AMR partitioners using a suite of "real-world" applications is presented in [7].

An application-centric characterization of the partitioners is obtained by first classifying the SAMR application state using the octant approach, and identifying the partitioning requirements of each octant. We then assign partitioner(s) to application state-octants based on their ability to meet the requirements of that octant. The associations of application state octants to partitioning techniques are summarized in Table 2. In the Pragma framework, this AMR application characterization and mapping of partitioners to application state define the polices for runtime adaptation.

Octant	Scheme
I	pBD-ISP, G-MISP+SP
II	pBD-ISP
III	G-MISP+SP, SP-ISP
IV	G-MISP+SP, SP-ISP, ISP
V	pBD-ISP
VI	pBD-ISP
VII	G-MISP+SP
VIII	G-MISP+SP, ISP

Table 2. Recommendations for mapping octants onto partitioning schemes.

4.5 Experimental Evaluation of the Adaptive Meta-Partitioner

The experimental evaluation of adaptive meta-partitioning uses RM3D, a 3-D compressible turbulence application solving the Richtmyer-Meshkov instability. Application characterization consists of two steps. First, the adaptive behavior of the application was captured in an adaptation trace generated using a single processor run. The adaptation trace contains snap-shots of the SAMR grid hierarchy at each regrid step. This trace was then analyzed using the octant approach and the adaptive partitioning strategy was defined [1]. The application characterization presented in this paper was performed manually. However, we are currently developing agent-based mechanisms for automatically performing the characterization at run-time.

In this experiment we used a base grid of size 128*32*32 and 3 levels of factor 2 space-time refinements. Regridding was performed every 4 time-steps at each level. The application ran for 800 coarse level time steps and the trace consisted of over 200 snap-shots. A selection of these snap-shots are shown in Fig. 3 and the octant-based characterization of the RM3D trace is summarized in Table 3.

The experimental study of the adaptive meta-partitioner was performed on the NPACI IBM SP2, *Blue Horizon*, at the San Diego Supercomputing Center. The experiments consisted of measuring application execution times for dif-

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

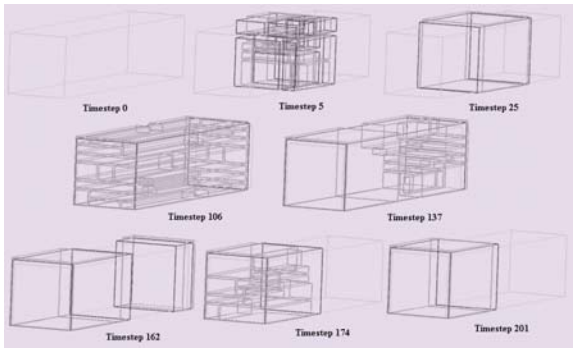

Figure 3. RM3D profile views at sampled time-steps.

Time-step	Octant State	Partitioner
0	IV	G-MISP+SP
5	VII	G-MISP+SP
25	I	pBD-ISP
106	VI	pBD-ISP
137	VIII	G-MISP+SP
162	II	pBD-ISP
174	V	pBD-ISP
201	III	G-MISP+SP

Table 3. Characterizing RM3D application run-time state for partitioning behavior.

ferent processor configurations, with the partitioning parameters switched on-the-fly during application execution.

Partitioner	Run-time (sec)	Max. Load Imbalance(%)	AMR Efficiency(%)
SFC	484.502	24.878	98.8207
G-MISP+SP	405.062	11.3178	98.7778
pBD-ISP	414.952	35.0317	98.8582
"adaptive"	352.824	8.11825	98.7633

Table 4. Partitioner performance for RM3D application on 64 processors.

The metric results for 64 processors are listed in Table 4. Dynamically switching partitioners results in reduced runtimes, with 27.2% improvement over the slowest partitioner for 64 processors. In the adaptive partitioner, G-MISP+SP is used to improve load-balance when the application is computationally dominated, while pBD-ISP reduces communication and data-migration overheads.

4.6 System Sensitive Adaptations

System sensitive adaptation uses system state information to appropriately configure the partitioners selected based on the application state. Current system parameters are obtained using NWS and are used to compute their relative computational capacities for the elements of the grid. The system-sensitive partitioner for dynamic distribution and load balancing then uses these relative capacities. For example, let us assume K elements in computational grid. The relative capacity C_K for the k^{th} grid-element is defined as the weighted sum of normalized values of the individual available CPU P_K, memory M_K, and link bandwidth L_K capacities returned by NWS. Weights are application dependent and reflect its computational, memory, and communication requirements. Once the relative capacities of the processors are computed, the workload is distributed proportionately among them. The overall operation is shown in Figure 4. Note that this discussion simply illustrates the concept and does not make use of performance prediction capabilities of the analysis module discussed above or the agent based management capabilities.

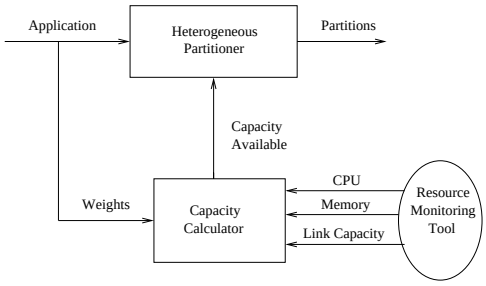

Figure 4. System sensitive adaptive AMR partitioning.

The system sensitive adaptive partitioner is evaluated using the RM3D compressible turbulence kernel executing on a Linux-based workstation cluster. The kernel 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 (for simulating heterogeneous loads on the cluster nodes) 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 partitioning scheme. This latter scheme performs an equal distribution of the workload on the processors.

The percentage improvement achieved using the system sensitive partitioner over the default partitioner is listed in Table 5. Relative capacities of the processors are calculated only once before the start of the simulation in this experiment. 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. Details about system sensitive adaptations are available in [6].

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**

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

Table 5. Improvement due to system-sensitive adaptive partitioning.

4.7 Agent-based Automated Adaptations in Pragma

In the discussion above, we have illustrated the application and system characterization and the adaptive application runtime management using the system and application adaptive partitioning as the example. Using application management agents and the predictive system characterization models, Pragma extends this process to adaptively manage all applications components in an automated, scalable, reliable, and efficient manner. For example, in the case study, CATALINA agents resident at each computing element in the distributed environment can be programmed to monitor the current state of each processing element. Note that the agents use information provided by the monitoring system such as NWS and provide an application specific semantic interpretation of this raw data. Local state information is published to the message-center allowing each agent to have direct and immediate access to all relevant information to drive local adaptations. Furthermore, each local agent can be programmed to monitor and publish application related events. For example, a local agent is used to generate events when the load reaches a certain threshold - this event can then trigger repartitioning. Similarly, a change in the effective communication bandwidth can trigger a similar repartitioning coupled with a selection of a partitioner and communication mechanism that can tolerate the increased communication latency. The local agent uses available state information, predictive performance functions and the policy database to make local inferences, for e.g. that local load needs to decrease by X% or the communication/computation ratio for local assignments has to be improved by Y%. Local decisions are hierarchically consolidated by the application delegation manager agent (ADM). This agent initiates changes in the system configurations or requests additional resources as required. Final policy decisions are then propagated to the individual local agents. Note that each local agent is autonomous in that it does not need to participate in the initial inference - the only requirement is that the AMD recommendations be complied with.

5 Conclusions

In this paper we presented an overview of Pragma, an adaptive runtime infrastructure 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 overarching motivation for this research is to enable the next generation of very large-scale dynamically adaptive scientific and engineering simulations on widely distributed and highly heterogeneous and dynamic execution environments such as the computational "grid". Pragma combines 4 key components: system characterization and abstraction component, application characterization component, active network control, and policy-base. The design of Pragma is driven by three astrophysical simulations chosen to be representative of a wide variety of important simulations and to expose many of the problems presently encountered (and currently unsolved) by computational physicists. The design, prototype implementation and preliminary evaluations of Pragma components were presented.

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] S. Hariri and et al. CATALINA: A Smart Application Control and Management. submitted to the *Active Middleware Services Conference*, 2000.

[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 Sci. on Engineering Complex Computer Systems, 1999.

[4] 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.

[5] 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.

[6] S. Sinha and M. Parashar. Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters. *3rd IEEE Intl. Conference on Cluster Computing*, 435-442, 2001.

[7] J. Steensland, S. Chandra, M. Thune, and M. Parashar. Characterization of Domain-based Partitioners for Parallel SAMR Applications. *PDCS*, 425-430, Nov. 2000.

[8] J.S. Vetter and K. Schwan. Optimizations for language-directed computational steering. *IPPS 99*, 1999.

[9] R. Wolski. Forecasting Network Performance to Support Dynamic Scheduling using the Network Weather Service. *6th IEEE Symp. on HPDC*, 1997.

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 **IEEE**