Using Cross-Layer Adaptations for Dynamic Data Management in Large Scale Coupled Scientific Workflows

Tong Jin, Fan Zhang, Qian Sun, Hoang Bui, Manish Parashar
NSF Cloud and Autonomic Computing Cente
Rutgers Discovery Informatics Institute
Rutgers University, Piscataway, NJ 08854, USA
{tjin, zhangfan, qiansun, hbui, parashar}@cac.rutgers.edu

Hongfeng Yu
Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588, USA
yu@cse.unl.edu

Scott Klasky, Norbert Podhorszki, Hasan Abbasi
Oak Ridge National Labortory
P.O. Box 2008, Oak Ridge, TN, 37831, USA
{klasky,pnorbert,habbasi}@ornl.gov

ABSTRACT

As system scales and application complexity grow, managing and processing simulation data has become a significant challenge. While recent approaches based on data staging and in-situ/in-transit data processing are promising, dynamic data volumes and distributions,such as those occurring in AMR-based simulations, make the efficient use of these techniques challenging. In this paper we propose cross-layer adaptations that address these challenges and respond at runtime to dynamic data management requirements. Specifically we explore (1) adaptations of the spatial resolution at which the data is processed, (2) dynamic placement and scheduling of data processing kernels, and (3) dynamic allocation of in-transit resources. We also exploit coordinated approaches that dynamically combine these adaptations at the different layers. We evaluate the performance of our adaptive cross-layer management approach on the Intrepid IBM-BlueGene/P and Titan Cray-XK7 systems using Chombo-based AMR applications, and demonstrate its effectiveness in improving overall time-to-solution and increasing resource efficiency.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; C.2.4 [**Computer - Communication Networks**]: Distributed Systems; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

Keywords

Cross-layer adaptation, in-situ/in-transit, coupled simulation workflows, staging, data management

1. INTRODUCTION

Advanced coupled simulation workflows running at extreme scales are providing new capabilities and new opportunities for insights in a wide range of application areas. These workflows compose multiple physical models and codes along with data processing and analysis services, and are presenting new challenges due to their scales, coupling and coordination behaviors and overall complexities, which must be addressed before their potential can be fully realized. For example, many of these simulations are based on dynamically adaptive formulations such as Adaptive Mesh Refinement (AMR), which exhibit dynamic runtime behaviors and result in large and dynamically changing volumes of data. Efficiently managing, transporting and analyzing this data have become significant and immediate challenges.

Recent approaches based on data staging [5, 6, 16] and in-situ/in-transit data processing [25, 3] that are attempting to address these challenges are promising – these approaches offload data processing to separate resources on the same systems (in-transit) and/or perform the processing directly on the resources that are running the simulation (in-situ). For example, our previous work [3] demonstrated how a simulation plus analytic workflow can efficiently run on current high-end computing systems using a hybrid in-situ/in-transit approach. Specifically, we proposed to decompose the analytic components of the workflow into pieces that can scalably run in-situ, and pieces that must be executed in-transit – e.g., the raw data is down-sampled in-situ using a predefined sampling rate and is then transported to in-transit resources for further analysis. The effectiveness of this approach clearly depends on the mapping of workflow components, the size and distribution of the data and the resources available in-situ and in-transit, and achieving efficiency and scalability requires carefully configuring the staging resources and the mappings based on application behaviors. While these can be pre-configured for relatively simple and static workflows, such an approach becomes ineffective when application behaviors become dynamic, as is the case for AMR-based simulations – in AMR-based simulations, dynamic refinements can lead to imbalanced data distributions and heterogeneous resource (memory, CPU, network bandwidth) requirements.

In this paper we explore a cross-layer adaptive runtime approach to address these challenges. Specifically, we explore runtime adaptations at three different layers, viz., application layer, middleware layer, and resource layer. We implement these adaptations as part of an autonomic runtime and evaluate their abilities to respond to dynamic data processing requirements and resource constraints for coupled AMR-based simulation workflows. At the application layer, we dynamically adapt the spatial and temporal resolution of the data being written and processed; at the middleware layer, we adapt the in-situ and/or in-transit placement and scheduling of data processing operations; and at the resources layer we adapt the allocation of in-transit resources. We also explore a coordinated management approach that combines these adaptations in a cross-layer manner to further optimize the end-to-end performance of the workflow and to address requirements or constrains that cannot be effectively satisfied by adaptations at one layer alone.

We have deployed the adaptive cross-layer management runtime on the Intrepid IBM BlueGene/P system at Argonne National Laboratory and the Titan Cray-XK7 system at Oak Ridge National Laboratory. We use these deployment along with the Chombo [1]-based AMR simulations plus data visualization workflow to experimentally evaluate the behavior and performance of the individual adaptations at each layer, as well as the effectiveness of the dynamic and coordinated cross-layer approach in improving overall time-to-solution, increasing resource efficiency, and mitigating I/O costs.

The rest of this paper is organized as follows. Section 2 presents the data management challenges of advanced coupled simulation workflows and highlights the challenges of dynamic, AMR-based simulation plus data visualization workflows. Section 3 describes the conceptual architecture and operation of the adaptive cross-layer management runtime and its components. Section 4 develops specific adaptation policies at the 3 layers and the combined cross-layer adaptation. Section 5 presents the results of our experiments using the Chombo-based application workflow on Intrepid and Titan. Section 6 discusses related work. Section 7 concludes the paper and outlines future work.

2. PROBLEM DESCRIPTION

As noted in the introduction above, dynamically adaptive formulations of simulations, such as those based on Adaptive Mesh Refinement (AMR), exhibit dynamic runtime behaviors and result in large and dynamically changing volumes of data, imbalanced data distributions and heterogeneous resource (memory, CPU, network bandwidth) requirements. To illustrate the dynamic data management and processing requirements of AMR-based simulations, consider the *3-D AMR Polytropic Gas* application that is part of the Chombo package [1] developed by the Lawrence Berkeley National Laboratory. This application implements the Godunov unsplit algorithm for integrating systems of conservation laws (e.g., the Euler equations of gas dynamics). Figure 1 plots parts of a profile of the distribution of the application's peak memory usage on 4000 CPU cores over 50 time steps. As we can see from this plot, memory usage varies significantly,

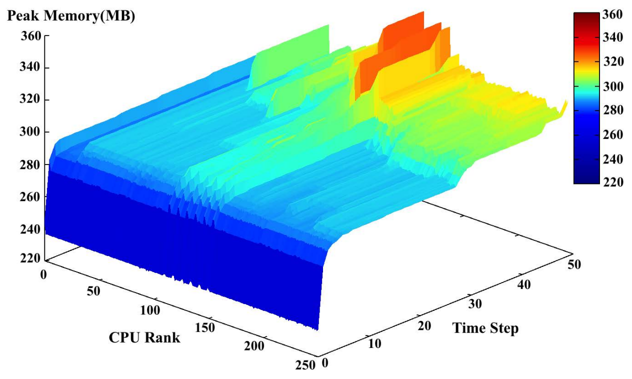


Figure 1: Distribution of the peak memory consumption for an AMR-based Polytropic Gas simulation using the Chombo library.

both across cores and over time. More importantly, the peak memory usage can be as high as several Gigabytes per node if multiple memory hungry process are placed in the same multi-cores node.

Such dynamic runtime behaviors of SAMR-based simulation workflows increase the complexity of managing and processing the data they produce, including managing the staging resources and scheduling in-situ and/or in-transit data processing while satisfying constraints on the amount of data movement, the overhead on the simulation, and/or the level of analytics. For example, AMR-based simulations involve dynamic local refinements, which can significantly increase the resources consumed by the simulation on a subset of nodes. This in turn reduces the resources available for in-situ analytics. At the same time, it also increases the spatial and temporal resolutions of data, and correspondingly the computational and storage requirements for the analytics as well as the cost of data movement if the analytics have to be executed in-transit. The increasing computational and storage requirements of the analytics can also impact in-transit resource requirements. Note that as the simulations evolves, refined regions maybe further refined or coarsened, which can result in different sets of requirements and constraints.

Clearly, making staging and in-situ/in-transit processing approaches effective for these dynamic applications given performance, overhead and resource constraints requires runtime adaptations and tradeoffs. Furthermore, these adaptations may be explored at different levels. At the application level, the application may be able to adapt the spatial and/or temporal resolution of the analytics or limit the analytics to "interesting" regions, to meet constraints on the type of analytics, the available resources, and/or acceptable overheads. Similarly, at the runtime level, the placement and scheduling of in-situ/in-transit tasks can be adapted, and at the resource level, the number of in-transit resources can be adapted. In this paper we explore how we can realize these dynamic adaptations at runtime for AMR-based simulation workflows on large-scale systems. We also explore policies and mechanisms for combining these adaptations in a coordinated and cross-layer manner to better address ap-

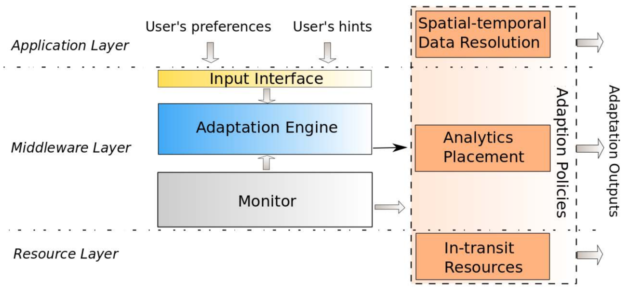


Figure 2: A conceptual architecture of an autonomic approach for realizing runtime adaptations for in-situ/in-transit implementations of coupled simulation workflows.

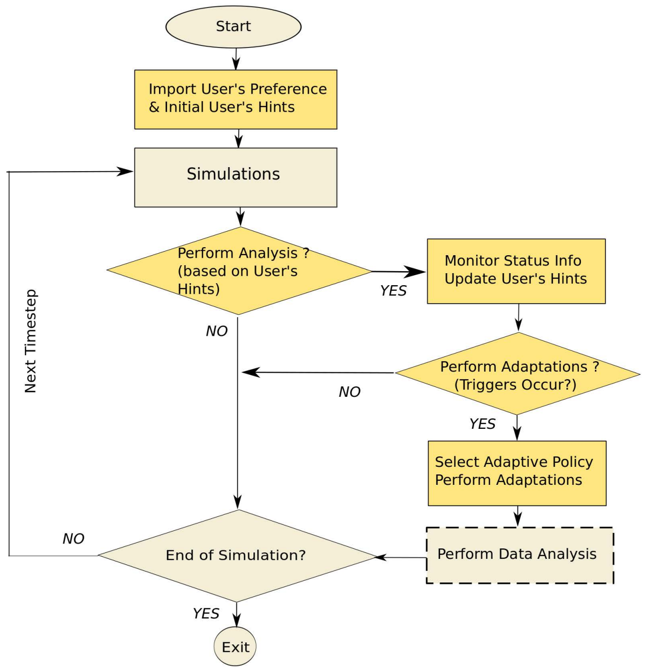
plication requirements and constraints.

3. REALIZING CROSS-LAYER ADAPTATIONS FOR LARGE-SCALE SIMULATION WORKFLOWS

This section describes our approach for efficiently and scalably realizing runtime adaptations for in-situ/in-transit implementations of coupled simulation workflows. The conceptual architecture follows an autonomic approach and consists of three key components, a monitor, the adaptation engine, and adaptation policies, as illustrated in Figure 2 and described below. In our approach, users can provide two types of inputs. *User preferences* define the objectives that users expect to achieve, such as minimizing time-to-solution, minimizing data movement, using highest available data resolution, etc. *User hints* provide additional information to the adaptation engine based on the user's knowledge of the application workflow and on past experience, for example, tolerance to data downsampling, nature of regions of interest, possible adaptation phases and/or patterns, etc.

The *Monitor* captures runtime status information at the different layers, i.e., application, middleware, and resource, and uses it to characterize the current operational state of the system and application and trigger adaptations if appropriate. Status information includes resource utilization and resource availability (memory, bandwidth, CPU cores) as well as application execution time, analysis time and the size of the generated data. The *Adaptation Engine* is responsible for selecting and executing appropriate adaptations based on user preference and hints, the operational state provided by the monitor, and adaptation policies.

Three adaptation mechanisms are explored in this paper. In the first mechanism, the application layer changes the spatial and/or temporal resolution of data generated in-situ before its is moved to the in-transit resources for processing. This mechanism can adjust the frequency of in-situ data reduction as well as the type of reduction performed by appropriately selecting the parameters of the data reduction module (e.g., down-sample factor, compression rate, etc.). The second adaptive mechanism adapts the placement of the data processing operations at middleware layer. Placements can be in-situ, in-transit or hybrid (i.e., in-situ + in-transit). The third adaptation mechanism targets the resource layer. It determines the number of in-transit resources needed and dynamically allocates resources for in-transit processing if



Start
Import User's Preference & Initial User's Hints
Simulations
Perform Analysis ? (based on User's Hints)
YES
Monitor Status Info Update User's Hints
NO
Next Timestep
Perform Adaptations ? (Triggers Occur?)
NO
YES
Select Adaptive Policy Perform Adaptations
NO
End of Simulation?
Perform Data Analysis
YES
Exit

Figure 3: An overview of the autonomic adaptation process.

necessary.

The *Adaptation Policies* specify which adaptation mechanism(s) should be executed based on user inputs and the operational state. In the following Section, we develop adaptation policies at each of the layers as well as a policy for combined cross-layer adaptation.

The overall adaption process is illustrated in Figure 3. The operational status of the simulation workflow is periodically (e.g., after every specified number of simulation time steps) sampled by the*Monitor* and forwarded to the *Adaptation Engine*, which determines if an adaptation is required and triggers the appropriate adaptation(s).

4. DEFINING ADAPTATION POLICIES

In this section, we develop adaptation policies for an AMR-based simulation workflow. Note that rather than finding optimal adaptations, our goal is to develop policies that can be efficiently and scalably implemented at runtime on very large scale system while satisfactorily addressing application requirements/constraints. Specifically, we develop policies for each of the 3 layers as well as a cross-layer policy of coordinated adaptations, which are described in the following subsections. Table 1 summarizes the notation used in this discussion.

4.1 Policy for Adaptation at the Application Layer

The application layer adaptive mechanism controls the resolution of the data that is forwarded to the analysis methods, and enables a trade-off between the time and resources spent on analysis and the resolution at which the analysis is performed. For example, it may be beneficial to have some analysis done even if it is performed at a lower resolution. The goal of this adaptation is to determine the data resolu-

Symbol	Description	Equations numbers where used
S_{data}	size of simulation output	(1)(8)(10)
X	down-sampling factor	(1)(3)
$f_{data\_reduce}(S_{data}, X)$	data reduction operation	(2)
$Mem_{data\_reduce}(S_{data}, X)$	memory needed to perform data reduction	(2)
$Mem_{available}$	total available memory	(2)
T_{sum_insitu}	total wallclock time at in-situ resources	(4)(6)
$T_{sum\_intransit}$	total wallclock time at in-transit resources	(5)(6)
N	number of simulation processors	(4)
M	number of in-transit processors	(5)
$ITER$	total number of iterations	(4)
D_i	final decision for executing analysis: 1 for in-situ, 0 for in-transit	(4)(5)(7)(8)
$T_{i\_sim}(N)$	execution time of the ith iteration of the simulation	(4)(9)
$T_{i\_insitu}(N, S_{i\_data})$	execution time for the ith in-situ analysis on N processors	(4)(7)
$T_{i\_intransit}(M, S_{i\_data})$	execution time for the ith in-transit analysis on M processors	(5)(9)
$T_{i\_intransit\_wait}$	idle time on the in-transit side	(5)
$T_{i\_insitu\_wait}$	idle time on the in-situ side	(4)
$T_{j\_intransit\_remaining}(M, S_{j\_data})$	remaining execution time for the jth in-transit processing iteration	(7)
$Mem_{insitu}(S_{i\_data}, N)$	memory cost for in-situ processing	(8)
$Mem_{intransit}(S_{i\_data}, M)$	memory cost for in-transit processing	(8)(10)
$T_{i\_sd}(S_{data})$	latency associated with sending data	(9)
$T_{i\_recv}(S_{data})$	latency associated with receiving data	(9)

Table 1: Notation used in formulating adaptation policies.

tion that can be effectively processed in-situ or transferred to in-transit resources given user preferences and the current operational state. Specifically, it determines the factor (X) by which to downsample the simulation data. This is either selected from a set of acceptable downsampling factors provided by the user as a hint, or generated automatically based on information content of interest. The selection is made based on the available memory and the memory needed to implement downsampling factor X, and the smallest value of X that can be used given the memory constrains is selected. The downsampling factor for the i^{th} simulation iteration is determined by the following policy:

Maximize

$$S_{data} - f_{data\_reduce}(S_{data}, X) \tag{1}$$

Subject to

$$Mem_{data\_reduce}(S_{data}, X) \leqslant Mem_{available} \tag{2}$$

(memory requirement)

$$when X \epsilon \{X_1, X_2, \cdots, X_n\} \tag{3}$$

(set of acceptable down-sample factors)

4.2 Policy for Adaptation at the Middleware Layer

Adaptations at the middleware layer target the placement of the analytics, in-situ, in-transit or hybrid, to minimize the overall time-to-solution under the current resource constraints. The policy considers three cases: (1) If there are sufficient memory resources to perform the analysis either in-transit or in-situ but not both, the adaptation will place the analysis at the location where the memory resources are available. (2) If there are sufficient memory resources at both locations and in-transit CPU resources are available, the analysis will be placed in-transit since the analysis can run in parallel with the simulation. (3) If the in-transit cores are busy processing simulation data generated at previous time steps, the adaptation engine will estimate the remaining time for such in-transit data processing as well as the execution time if the current data is processed in-situ. If

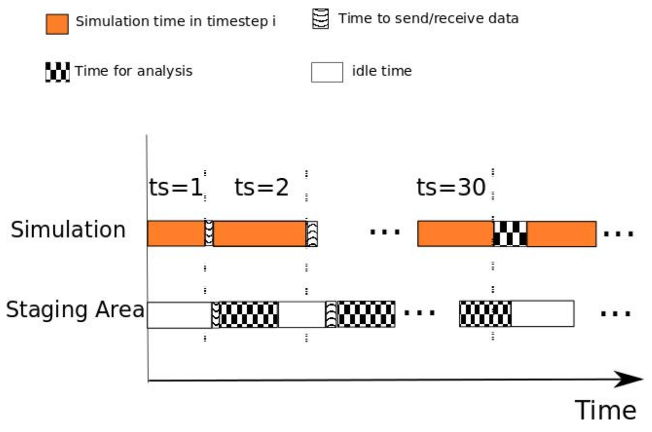


Figure 4: Illustration of the analysis placement adaptation policy. For adaptation at ts=1 and 2, in-transit resources are idle, and as a result analysis is placed in-transit. For adaptations at ts=30, since in-transit resources are busy, the analysis time for in-situ and in-transit processing are estimated, and the analysis is placed in-situ is the estimated processing time there is shorter. Note that the data transfer is asynchronous and its is assumed that the effective time for transferring the data is much smaller than the time for processing the data.

the in-situ data processing is estimated to be faster, the analysis will be performed in-situ. Otherwise, the data will be asynchronously transferred to the in-transit nodes and will be processed as soon as in-transit cores become available. These latter two cases are illustrated in Figure 4 and are expressed in the formulations below:

Since

$$T_{sum\_insitu} \simeq \sum_{i=1}^{ITER}\{T_{i\_sim}(N) + D_i \cdot (T_{i\_insitu}(N, S_{i\_data}))$$

$$+ \bar{D}_i \cdot (T_{i\_insitu\_wait})\} \tag{4}$$

$$T_{sum\_intransit} \simeq \sum_{i=1}^{ITER}\{\bar{D}_i \cdot T_{i\_intransit}(M, S_{i\_data}) + T_{i\_intransit\_wait}\} \quad (5)$$

Minimize

$$max\{T_{sum\_insitu}, T_{sum\_intransit}\} \quad (6)$$

(minimized time-to-solution)

Subject to

$$\bar{D}_i \cdot (T_{j\_intransit\_remaining}(M, S_{j\_data}) < T_{i\_insitu}(N, S_{i\_data})) = 1, j < i; \quad (7)$$

(execution time estimation)

$$D_i \cdot (Mem_{intransit}(S_{i\_data}, M) < S_{data}) + \bar{D}_i \cdot (Mem_{available} \leqslant Mem_{insitu}(S_{i\_data}, N)) = 1 \quad (8)$$

(resource constraints).

4.3 Policy for Adaptation at the Resource Layer

Performing analysis in-transit minimizes its impacts on the simulation and can achieve better time-to-solution. However, this approach reduces the computational resource available to the simulation, which in turn can offset this advantage.

The resource layer adaptation targets this trade-off between minimizing the impacts of analysis on the simulation (i.e., improving time-to-solution) and minimizing the resources used for in-transit processing. For in-transit processing, the ideal time-to-solution can be achieved if in-transit analysis on simulation data generated during the ith time step finishes before data from the $(i + 1)$th simulation time step is ready to be sent. In other words, the smaller the idle time at the in-transit resources, the more efficiently the in-transit resources are utilized. On the other hand, sufficient in-transit resources are needed to cache the simulation data generated at current time step. Therefore, the resource layer adaptation first determines the minimum number of in-transit cores required based on the size of the simulation data and the required in-transit memory resources. It then estimates the in-transit processing time, and if this time is greater than the time required for a simulation time step, the number of in-transit cores is increased until the ideal in-transit processing time is achieved, i.e., time for the in-transit analysis is less than the time for a simulation time step and the in-transit idle time is minimized. This policy is expressed in the formulations below:

Minimize M

Subject to

$$T_{i+1\_sim}(N) + T_{i+1\_sd}(S_{i+1\_data}) = T_{i\_intransit}(M, S_{data}) + T_{i\_recv}(S_{i\_data}) \quad (9)$$

(Expected same execution time on both simulation side and in-transit side)

and

$$Mem_{intransit} > S_{data} \quad (10)$$

(in-transit memory constraint)

4.4 Policy for Combined Cross-Layer Adaptation

The cross-layer adaptation policy explores the coordinated use of the adaptive mechanisms at the three layers described above to satisfy user objective or constraints, e.g., for desired time-to-solution or acceptable data movement, that cannot be satisfied using adaptations at one layer alone, or to further improve performance. Specifically, we design a heuristic *root-leaf* policy for the selection of adaptation mechanisms across the three layers. This policy consists of three steps: selecting *root mechanisms*, selecting *leaf mechanisms*, and executing selected mechanisms. Consider "minimizing time-to-solution" as an example objective to illustrate these steps of the policy. First, the policy selects the mechanisms that can address the objectives of the cross-layer adaptation, and marks them as *root mechanisms.* Based on the descriptions of adaptation mechanisms above, the middleware layer adaptation can address our example object of minimizing time-to-solution and should be automatically included in the set of root mechanisms. Second, the policy goes through the formulation of *root mechanisms* and looks for data dependencies with mechanisms at other layers not in the set. In our example, the data size S_{i_data} and the number of in-transit cores M are two significant inputs for *root mechanism*, i.e., the middleware layer adaptation mechanism, and these parameters also impacted by the application layer adaptation mechanism for data reduction and resource layer adaptation mechanism. Therefore, these mechanisms are marked as *leaf mechanisms.* Finally, once both *root mechanisms* and *leaf mechanisms* are selected, the policy executes these adaptations, first the *leaf mechanisms* and then the *root mechanisms.* If there are data dependencies among the selected *leaf mechanisms*, the execution is in the order of the dependencies, i.e, *leaf mechanisms* that do not rely on outputs from the other selected mechanisms are executed first, followed by the mechanisms that depend on their outputs. In our example, the application layer adaptation will be executed first since its output S_{i_data} will impact the resource layer adaptation mechanism, i.e., the other selected *leaf mechanism.* The middleware adaptation mechanism will be executed last as it is the *root mechanism* in our example.

Similarly, if the user-defined objective is to maximize in-transit resource utilization, the policy would select resource layer adaptation as the *root mechanism* and the application layer adaptation as the *leaf mechanism.* The middleware layer adaptation mechanism will not be included in this case since it has no data dependency with the *root mechanism.*

5. EXPERIMENTAL EVALUATION

In this section we present an experimental evaluation of the adaptive runtime management framework presented in this paper. We first evaluate adaptations at each of the three layers individually, and then evaluate combined cross-layer adaptations.

5.1 Experiment Setup

AMR-based Simulation Workflow.

The evaluation presented in this section uses a simulation workflow that is composed of a Chombo [1]-based AMR simulation and a visualization service, which are described below.

Chombo-based AMR Simulations: We use two different AMR-based simulations that are distributed as part of the Chombo AMR package [1]. Both these simulations implement the AMR Godunov unsplit algorithm but show very different performance characteristics. The *AMR Advection-Diffusion* simulation implements an adaptive conservative transport (advection-diffusion) solver, while the *AMR Polytropic Gas* implements the AMR Godunov unsplit algorithm for integrating systems of conservation laws (e.g., the Euler equations of gas dynamics). While both simulations exhibit runtime adaptations, the latter is more memory and compute intensive, especially in 3-D.

Visualization Service: The visualization service implements the marching cubes algorithm [13, 21], the *de facto* standard isosurface extraction algorithm in scientific visualization, to construct triangular meshes from AMR data according to user specified isovalues. The algorithm scans each cell and conducts triangulation depending only on the values of the current cell, and thus the isosurface construction is performed locally. The ghost regions are managed by Chombo, and there is nearly no communication needed for the marching cubes algorithm. Using this algorithm we can extract isosurfaces from full-resolution data in-situ, which can generate a high-quality triangular mesh to capture the fine structural information.

Implementation of the Adaptive Runtime.

The adaptive runtime is implemented on the top of our DataSpaces data-management substrate [3, 6, 5]. DataSpaces provides distributed interaction and coordination services to support in-situ and in-transit simulation-analysis workflows on very large-scale systems, and its data transport layer provides the required asynchronous communication and data transfer services.The *Adaptation Engine* is integrated with DataSpace to enable runtime coordination and adaptation at different the layers. In addition, the embedded performance tools within Chombo provides runtime system information such as memory usage and execution time, and are used by the *Monitor*.

Systems.

Our experiments were conducted on the Intrepid IBM BlueGene/P system at Argonne National Laboratory and Titan Cray-XK7 systems at Oak Ridge National Laboratory. Intrepid consists of totally 40960 nodes, each of which has quad-core processor and 2GB RAM (i.e., 500MB per core). Its peak performance can reach 557 teraflops.

Titan has 18,688 nodes connected through the Gemini internal interconnect, and each node has a single 16-core AMD 6200 series Opteron processor and 32GB RAM (i.e., 2GB per core). The total system memory is 600 terabytes and the system peak performance can reach 20 petaflops. Besides, it has 18688 K20 Keplers GPUs, although We didn't use them in our experiments.

5.2 Evaluation and Discussion

5.2.1 Evaluation of Adaptations at the Application Layer

For these experiments we used the memory intensive 3-D AMR Polytropic Gas application with a domain size of $128 \times 64 \times 64$ at the base level. The experiments were performed on 4K cores of the Intrepid IBM BGP system, which has only 500MB of memory per core. Furthermore, we experimented with two different types downsampling approaches that can be used by the application layer adaptation mechanism as described below.

User-defined range-based data downsampling: In this experiment, the application layer adaptation mechanism used an in-situ downsampling method with user-defined ranges of down-sampling factors. These ranges of acceptable downsampling factors were specified as user hints, and were $\{2, 4\}$ for the first half of the simulation, and $\{2, 4, 8, 16\}$ for the second half.

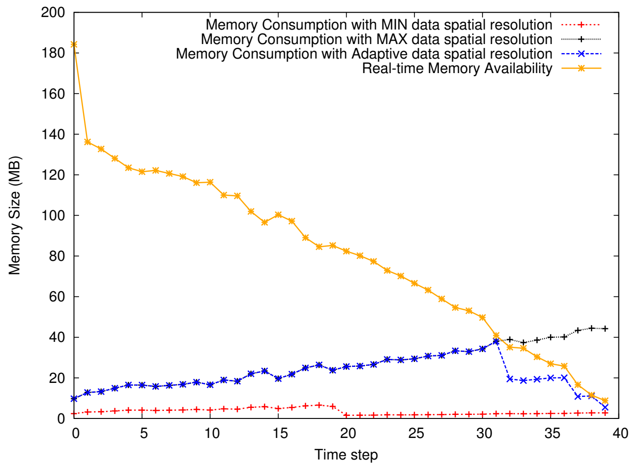


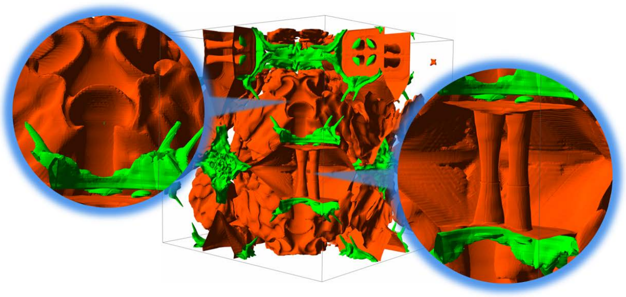
Figure 5: Evaluation of application layer adaptation of the spatial resolution of the data using user-defined downsampling factors, and based on runtime memory availability. Note that at the 31st time step, the spatial data resolution is reduced due to limited availability of memory resources, and at the 40th time step, the data resolution reaches the minimal value.

In this experiment, the peak memory used on a processor varied from $20MB$ to $>$ 300MB. Figure 5 plots the online memory availability for a single processor over 40 time steps. The Figure also shows the actual memory usage during the same period when using adaptive downsampling factors, as well as the memory requirements when maximum and minimum acceptable spatial resolutions were used for the data. When sufficient memory was available (between time step 0 to 30), the adaptive mechanism correctly selected the minimum down-sampling factor, which produced a larger data volume at a higher spatial resolution. However, starting at the 31st time step, the available memory could no longer support this higher spatial resolution. As a result, the adaptive mechanism increased the downsampling factor as seen in the figure.

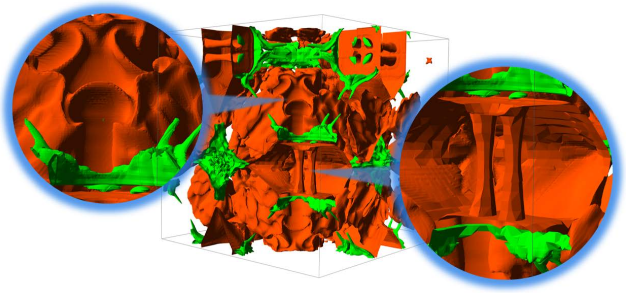
Entropy based data down-sampling: In these experiments, the down-sampling factors used by the application layer adaptation mechanism were automatically tuned based on information theory, which provided us with a theoretical framework to measure the information content of a variable [20]. For each data block within the AMR dataset, we compute the *entropy* value to quantify the distribution of its variables. For a discrete random variable χ and probability mass function $p(x), x \in \chi$, the entropy of X can be defined as

$$H(X) = -\sum_{x \in \chi} p(x) logp(x) \qquad (11)$$

where $p(x) \in [0, 1]$, $\sum_{x \in \chi} p(x) = 1.0$, and $-logp(x)$ represents the information associated with a single occurrence



(a)



(b)

Figure 6: Evaluation of application layer adaptation of the spatial resolution of the data using entropy based data down-sampling. (a) shows a simultaneous rendering of two isosurfaces of the full-resolution Polytropic Gas simulation data set. The surfaces are extracted from the density variable at the 60th time step, corresponding the isovalues 1.23 (red) and 4.18 (green), respectively. The right and left images show close up views of the two regions. (b) shows the result after the dynamic adaptation of its spatial resolution. The right region has its entropy value (at 5.14) that is lower than the specified threshold and thus is down-sampled at every 4th grid point. The left region has a higher entropy value (at 9.21) and its resolution is not changed.

of x. The higher the value of $H(x)$, the more information the data block contains. The unit of $H(X)$ is a *bit*. For example, at the 60th time step for the Polytropic Gas case, the entropy values of the data blocks at the finest level are between 5.14 and 9.85. We can now adaptively downsample the data blocks based on their entropy values by specifying a set of thresholds. Figure 6 compares visualizations using the full-resolution data and the adaptively down-sampled data. We can see that the fine structural information is well preserved for regions with higher entropy values, while regions with lower entropy values can potentially be reduced aggressively without losing much information or impacting our understanding of the data.

These results clearly show that our approach successfully adapts the down-sampling factor at runtime to meet the constraints on acceptable data resolution at the application layer as well as constraints due to the limited size of available memory at the resource layer. The results also show that such adaptations can potentially allow memory intensive simulation workflows to run on systems with contained memory resources.

5.2.2 *Evaluation of Adaptations at the Middleware Layer*

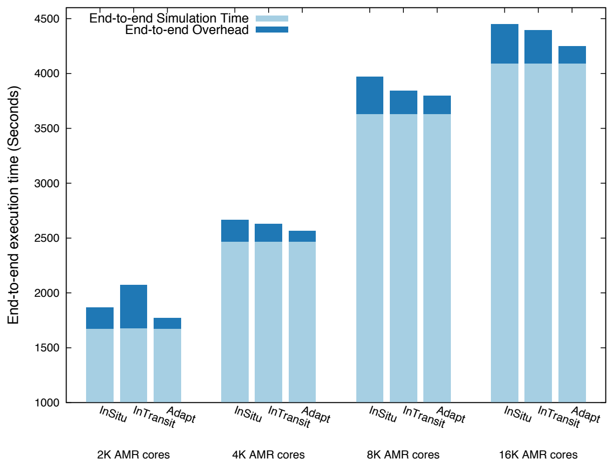


Figure 7: Comparison of cumulative end-to-end execution time between static and adaptive in-situ/in-transit placement of the visualization service. The end-to-end overhead plotted is the overhead on the overall time-to-solution and includes data processing time, data transfer time, and other system overheads.

In these experiments, we used the AMR Advection-Diffusion simulation and evaluated both, an adaptive placement and a static placement of the visualization service within the application workflow. The experiments were performed on Titan and evaluated how middleware layer adaptations can optimize overall time-to-solution at different scales. We ran the simulation on 2K, 4K, 8K and 16K cores, with a 16:1 ratio of the number of the simulation core to the number of the in-transit cores. The initial 3D grid domain sizes used in the experiments were $1024 \times 1024 \times 512$ for the 2K case, $1024 \times 1024 \times 1024$ for the 4K case, $2048 \times 1024 \times 1024$ for the 8K case, and $2048 \times 2048 \times 1024$ for the 16K case.

End-to-end execution time (or time-to-solution) was used as the key metric in our evaluation and is composed of two components as seen in Figure 7: *end-to-end simulation time* and *end-to-end overhead*. End-to-end overhead includes the data processing time, data transfer times and other system overheads such as due to adaptation. The adaptive in-situ/in-transit placement approach shows significant benefits as compared to a static approach in terms of the time-to-solution – it achieves the smallest cumulative end-to-end execution time, which demonstrates that our policy achieves it goal, i.e., to minimizes time-to-solution using adaptive placement. Quantitatively, the cumulative end-to-end execution overhead for the adaptive placement case decreased by 50.00%, 50.31%, 50.50%, 56.30% compared with static in-situ placement, and 75.42%, 38.78%, 21.29%, 48.22% as compared with static in-transit placement, for the 2K, 4K, 8K, and 16K core cases respectively. The end-to-end overhead in all these cases were less than 6% percent of the simulation time. Furthermore, since the analysis at some time steps were adaptively performed in-situ, the overall data movement for adaptive placement was reduced by 50.00%,

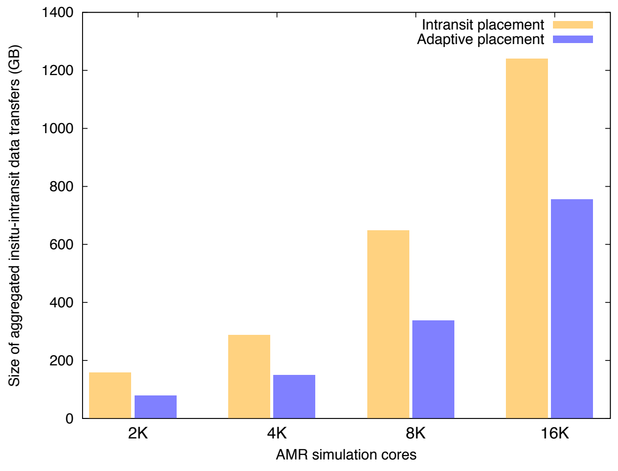


Figure 8: Comparison of total data movement between static and adaptive in-situ/in-transit placement of the visualization service.

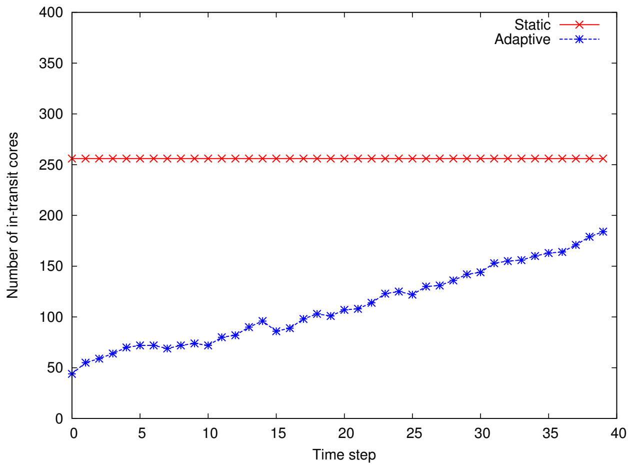


Figure 9: Number of in-transit processor cores performing analysis when using resource layer adaptations.

48.00%, 47.90%, 39.04% as compared to static placement for the 2K, 4K, 8K, and 16K core cases respectively, as shown in Figure 8.

5.2.3 *Evaluation of Adaptations at the Resource Layer*

In this experiment, we performed local adaptations at the resource layer to dynamically change the number of cores allocated for in-transit processing. With 4,096 simulation cores, the initial number of cores available as in-transit processing was 256. The rest of the setup for this experiment was the same as that described in Section 5.2.1. The goal of the experiment was to evaluate how effectively the resource layer mechanism respond to dynamic in-transit resource requirements while achieving efficient resource utilization.

Figure 9 plots the number of in-transit cores used at each time step. At the beginning of the simulation, the size of data generated and processed in-transit is relatively small. Therefore, only around 50 in-transit cores are needed. However, as the grid gets refined, the size of the data generated increases, and additional in-transit resources are required to satisfy memory requirement for in-transit analysis as well as time-to-solution constraints.

The adaptation approach uses fewer in-transit processor cores to achieve the same time-to-solution, compared with using a static number of in-transit cores. As a result, the resource utilization of the in-transit staging area is greatly improved. To quantify the improvement in CPU utilization, we define the cpu utilization efficiency as follows:

$$\frac{\sum_{j=1}^{TS}\sum_{i=1}^{M_j}\{T_{intransit\_analysis\_i\_j}\}}{\sum_{j=1}^{TS}\sum_{i=1}^{M_j}\{T_{intransit\_total\_i\_j}\}} \qquad (12)$$

where TS is the maximum time step, M_j is the number of in-transit cores allocated at the jth time step, $T_{intransit\_analysis\_i\_j}$ is the execution time at the ith in-transit cores for data analysis at the jth time step, $T_{intransit\_total\_i\_j}$ is the total execution time at the ith in-transit cores at the jth time step.

Using this definition of utilization efficiency, we find that the utilization efficiency when using resource layer adaptations is 87.11% as compared to only 54.57% in the static allocation case.

5.2.4 *Evaluation of Time-to-Solution Aware Cross-layer Adaptations*

Adaptations at a single layer may not meet the scientists' requirements in some scenarios. For example, the scientists often attempt to find abnormities in an AMR-based simulation by visualizing the output data on-the-fly. In these cases visualizing data with lower spatial resolution is often sufficient and is more efficient. Furthermore, this visualization must be performed in-situ and/or in-transit while satisfying constraints on the overheads on the simulation, the resources used, and the overall execution time. Achieving this requires coordinated adaptations at the application layer to adapt the data resolution, and the middleware layer to appropriately place the visualization.

This experiment evaluates such a combined cross-layer adaptation across multiple layers. The objective is defined as minimizing time-to-solution, and to facilitate comparison, the basic experiment setup is the same as that used for the experiments described in Section 5.2.2. The experiment also used the same acceptable user-defined data sampling rates that were used in the experiments described in Section 5.2.1, which were once again provided as user inputs for possible application layer adaptations.

The experiment results demonstrate that, in this case, adaptations at all three layers are triggered and execute in a coordinated manner. Figure 10 plots the values of overall cumulative end-to-end overhead, which decrease by 52.16%, 84.22%, 97.84%, 88.87% for the combined cross-layer adaptation cases (i.e., global adaptations) for the 2K, 4K, 8K and 16K core cases respectively, as compared to the corresponding middleware layer only adaptations (i.e., local adaptations) that were presented in Section 5.2.2. Since the data is reduced in-situ using downsampling, the time required for in-situ analysis and in-transit analysis decreases due to the decreasing data volume. On the other hand, faster in-transit

Cases	Total Time Steps	No. of Time Steps for Different Utilizations of the In-Transit Cores			
No. of Sim Cores : No. of Staging Cores		100% Cores	75% Cores	50% Cores	Less than 50% Cores
2K:128	27	25	2	-	-
4K:256	42	8	13	4	17
8K:512	49	4	23	22	-
16K:1024	41	10	12	10	9

Table 2: Actual utilization of in-transit cores while performing in-transit analysis.

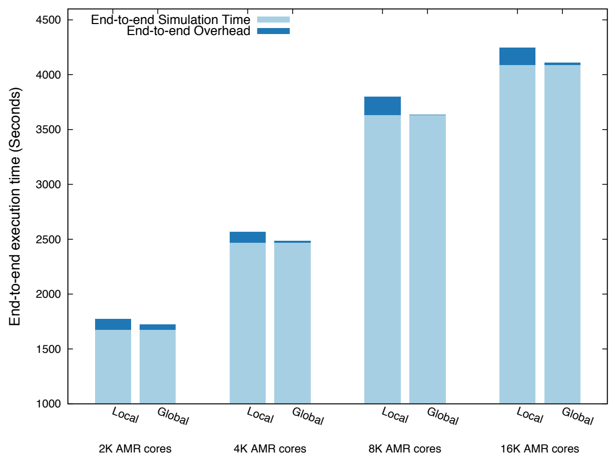


Figure 10: Comparison of cumulative end-to-end execution time between for combined cross-layer adaptations (i.e., global adaptations) and middleware layer only adaptations (i.e., local adaptations).

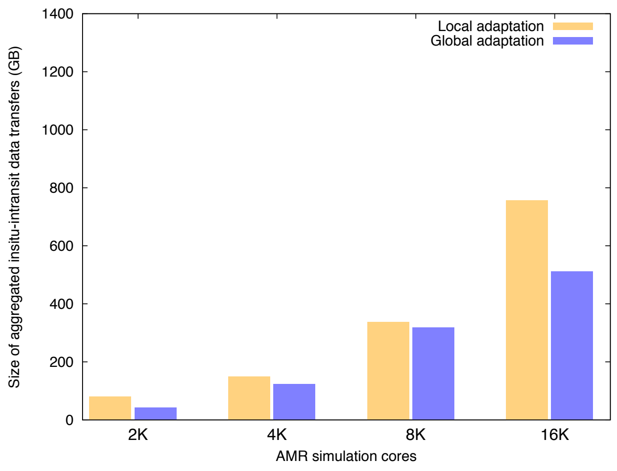


Figure 11: Comparison of the total data movement for combined cross-layer adaptations (i.e., global adaptations) and middleware layer only adaptations (i.e., local adaptations).

analysis implies that there is a greater possibility that in-transit resources will be idle between simulation time steps. In this case, middleware layer adaptations policy places the analysis in-transit more frequently, as demonstrated by the result in Table 2.

While performing more of the analysis in-transit implies a larger amount of data transfer, Figure 11 shows that the data reduction achieved due to the application layer adaptation dominates and the overall amount of data transfer actually decreased by 45.93%, 17.25%, 5.76%, and 32.41% or the 2K, 4K, 8K and 16K core cases respectively, as compared to the corresponding middleware layer only adaptations (i.e., local adaptations) that were presented in Section 5.2.2. Meanwhile, Table 2 shows that fewer in-transit cores are used to achieve the same time-to-solution, which demonstrates another benefit of this combined cross-layer adaptation. Specifically, for the 4K and 16K core cases, the fraction of the initially allocated in-transit cores used drops to less than 50% for some of the time steps.

In summary, our cross-layer adaptation approach can be triggered and can dynamically respond at runtime to meet user-defined objectives under varying resource limitation and user's constraints. Compared to static approaches, both the local adaptations at a single layer and global combined cross-layer adaptations demonstrate significant benefits in terms of time-to-solution, data movement, and resource utilization efficiency. Since our experiments use Chombo-based applications and a third-party adjustable visualization code, our cross-layer adaptive runtime can be used with other adaptive application frameworks, as well as other scalable analysis services with no/rare communications, such as descriptive statistic analysis, data subsetting, etc.

6. RELATED WORK

Simulation-time Data Processing: The increasing performance gap between computation and I/O in high-end computing environment is rendering traditional post-processing data analysis approaches based on disk I/O infeasible and inefficient. As a result, simulation-time data processing approaches have emerged, which operate on in-memory data before it is written to the disk or file systems. Several research projects have focused on two specific simulation-time analysis techniques, namely in-situ processing and in-transit processing.

In-situ data processing allows direct access to in-memory simulation data, and has been used in visualization [14], [22], [7], indexing building [10], data compression [11], multi-physics coupling [25], etc. This technique greatly reduces the costs due to data movement across the network because most data is available in local memory. However, due to the resource sharing nature of in-situ processing, it can increase the overall time-to-solution.

In-transit data processing executes data operations on dedicated compute resources in parallel with the simulations and thus minimizes the impact on the performance of the simulation and the overall time-to-solution. Many projects have studied the use of dedicated "staging" resources to support potential in-transit operations, such as DataStager [2], PreDatA [26], DataSpaces [5, 6]/ActiveSpaces [4], XpressSpace [24], GLEAN [19] and Nessie [15]. Our previous work also integrates a messaging system with data staging to support flexible data publish and subscribe patterns [9]. However, the data movement across the network in this approach can introduce large overheads as well as increase power consumption.

To take advantage of both in-situ and in-transit analytic placements, recent research [3] has explored the benefits of combining both in-situ and in-transit approaches on leadership class supercomputers, and demonstrated the importance of where the analytics execute in a hybrid in-situ/in-transit staging system. FlexIO [27] explored the trade-offs between performing analytics at different levels of the I/O hierarchy and supported a variety of simulation-analytics workloads through flexible placement options. However, these research efforts target static application workflows and preschedule the the placement of the analysis components. The adaptive placement supported by the cross-layer adaptive runtime framework presented in this paper can respond to the dynamic data management requirements of complex simulation analysis workflows based on adaptive formulations such as AMR, by scheduling the placement of analysis dynamically at runtime.

Single-Layer and Cross-Layer Adaptation: Previous research efforts have focused on improving performance by using a single-layer adaptive approach. Tapus et al. [18] introduced Resource Specification Language (RSL), a prototype language that performs adaptations by selecting appropriate program libraries and adaptively adjusting application parameters to tune the overall performance. This approach performs adaptations only at the application layer and does not adapt other layers. Similarly, Hsu et al. [8] proposed an algorithm that specifically targets the hardware layer, and automatically adapts CPU settings such as voltage and frequency to reduce power consumption in HPC environment.

Meanwhile, many researchers have noted that cross-layer adaptation can achieve performance improvements, especially when dealing with more complex workflows. For example, cross-layer adaptation methods can result in encouraging energy savings for mobile devices. Sachs et al. [17] employs a hierarchical approach that performs exhaustive global adaptation in conjunction with local adaptations. Although, at a smaller scale, they were able to achieve greater energy efficiency at four system layers: hardware layer, network layer, operating system layer, and application layer. Similarly, the GRACE-1 [23] framework, designed and implemented for mobile multimedia systems, supports application QoS under CPU and energy constraints via coordinated adaptation at the hardware, OS, and application layers. Moreover, the idea of cross-layer has been employed in gird computing environment to deal with the issues related to dynamic resource management [12].

However, the cross-layer adaptation approach has not been explored for dynamic simulation-analysis workflows on high-end systems. Our work proposes the cross-layer adaptation approach to enable dynamic adaptation for simulation-time data management and processing, and our large-scale experiments demonstrate the effectiveness in increasing resource efficiency and reducing overall time-to-solution on high-end HPC systems.

7. CONCLUSION AND FUTURE WORK

In this paper, we explored adaptive cross-layer adaptations to address the dynamic data volumes and data processing requirements of adaptive simulation workflows, such as those based on SAMR formulations. Specifically, we focussed on run-time adaptations across three different layers: application layer, middleware layer, and resource layer and demonstrated that such adaptations are necessary for meeting application requirements while meeting application and system constraints. Furthermore, we demonstrated that coordinated cross-layer adaptations can not only improve overall performance but can also enable the runtime to satisfy requirements and constraint that cannot be addressed by adaptations at a single layer alone.

The paper presented the design, implementation and evaluation of an autonomic cross-layer adaptation runtime composed of three key components: a monitor, the adaptation engine, and adaptation policies. The paper also presented adaptation policies at the different layers with corresponding triggers. The experimental evaluation presented results using AMR-based simulation codes implemented within the Chombo framework, and running on bot, the Intrepid IBM BlueGene/P system at ANL and the Titan Cray-XK7 system at ORNL. The evaluation results demonstrated the effectiveness of adaptation at each layer. The results also demonstrated that, in comparison to static approaches, the cross-layer approach with coordinated adaptations provides better performance, in terms of reducing network data movement, improving resource utilization and minimizing time-to-solution.

Our future work includes (1) designing and formalizing corresponding programming abstractions to support cross-layer adaptations and enable the user to express objectives and constraints and to formulate adaptation policies, (2) exploring adaptations involving data-placement across deep memory hierarchies, and (3) incorporating power/energy efficiency and power/performance trade-offs as part of the objectives.

8. ACKNOWLEDGMENTS

The research presented in this work is supported in part by the US National Science Foundation (NSF) via grants numbers ACI 1339036, ACI 1310283, DMS 1228203 and IIP 0758566, by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy through the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization (SDAV) under ward number DE-SC0007455, the Advanced Scientific Computing Research and Fusion Energy Sciences Partnership for Edge Physics Simulations (EPSI) under award number DE-FG02-06ER54857, the ExaCT Combustion Co-Design Center via subcontract number 4000110839 from UT Battelle, and by an IBM Faculty Award. The research was conducted as part of the NSF Cloud and Autonomic Computing (CAC) Center at Rutgers University and the Rutgers Discovery Informatics Institute (RDI2).

9. REFERENCES

[1] Chombo website, ”http://seesar.lbl.gov/anag/chombo”.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In *Proc. 18th International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009.

[3] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. W. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. PŐbay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proceedings of IEEE/ACM Supercomputing Conference (SC)*, November 2012.

[4] C. Docan, M. Parashar, J. Cummings, and S. Klasky. Moving the Code to the Data - Dynamic Code Deployment Using ActiveSpaces. In *Proc. 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*, May 2011.

[5] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. In *Proc. of 19th International Symposium on High Performance and Distributed Computing (HPDC'10)*, June 2010.

[6] C. Docan, M. Parashar, and S. Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing*, 15(2):163–181, 2012.

[7] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, October 2011.

[8] C.-H. Hsu and W. chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 1, nov. 2005.

[9] T. Jin, F. Zhang, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. A scalable messaging system for accelerating discovery from large scale scientific simulations. In *Proc. IEEE International Parallel and Distributed Processing Symposium (HiPC)*, December 2012.

[10] J. Kim, H. Abbasi, L. Chacon, C. Docan, S. Klasky, Q. Liu, N. Podhorszki, A. Shoshani, and K. Wu. Parallel in situ indexing for data-intensive computing. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 65 –72, oct. 2011.

[11] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. Samatova. Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1 –11, nov. 2011.

[12] C. Li and L. Li. Three-layer control policy for grid resource management. *J. Netw. Comput. Appl.*, 32(3):525–537, May 2009.

[13] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, Aug. 1987.

[14] K.-L. Ma. In Situ Visualization at Extreme Scale: Challenges and Opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19, 2009.

[15] R. Oldfield, P. Widener, A. Maccabe, L. Ward, and T. Kordenbrock. Efficient data-movement for lightweight i/o. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1 –9, sept. 2006.

[16] M. Parashar. Addressing the petascale data challenge using in-situ analytics. In *Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities*, PDAC '11, pages 35–36, New York, NY, USA, 2011. ACM.

[17] D. Sachs, S. Adve, and D. Jones. Cross-layer adaptive video coding to reduce energy on general-purpose processors. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 3, pages III–109. IEEE, 2003.

[18] C. Tapus, I.-H. Chung, and J. Hollingsworth. Active harmony: Towards automated performance tuning. In *Supercomputing, ACM/IEEE 2002 Conference*, page 44, nov. 2002.

[19] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 9 –14, oct. 2011.

[20] C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy*, 13:254–273, 2011.

[21] G. H. Weber, V. E. Beckner, H. Childs, T. J. Ligocki, M. Miller, B. van Straalen, and E. W. Bethel. Visualization of scalar adaptive mesh refinement data. *Numerical Modeling of Space Plasma Flows: Astronum-2007 (Astronomical Society of the Pacific Conference Series)*, 385:309–320, 2008.

[22] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma. In Situ Visualization for Large-Scale Combustion Simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, 2010.

[23] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, and R. H. Kravets. Grace-1: Cross-layer adaptation for multimedia quality and battery energy. *IEEE Transactions on Mobile Computing*, 5(7):799–815, July 2006.

[24] F. Zhang, C. Docan, H. Bui, M. Parashar, and S. Klasky. Xpressspace: a programming framework for coupling partitioned global address space simulation codes. *Concurrency and Computation: Practice and Experience*, 2013.

[25] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In *Proc. 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS'12)*, 2012.

[26] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on peta-scale machines. In *Proc. of 24th IEEE International Parallel and Distributed*

Processing Symposium (IPDPS'10), April 2010.

[27] F. Zheng, H. Zou, G. Eisnhauer, K. Schwan, M. Wolf, J. Dayal, T. A. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu. Flexio: I/o middleware for location-flexible scientific data analytics. 2013.