# AUTONOMIC DATA MANAGEMENT FOR EXTREME SCALE COUPLED SCIENTIFIC WORKFLOWS

**By**

**TONG JIN**

**A dissertation submitted to the**

**Graduate School—New Brunswick**

**Rutgers, The State University of New Jersey**

**In partial fulfillment of the requirements**

**For the degree of**

**Doctor of Philosophy**

**Graduate Program in Electrical and Computer Engineering**

**Written under the direction of**

**Manish Parashar**

**And approved by**

_____

_____

_____

_____

**New Brunswick, New Jersey**

**January, 2016**

## ABSTRACT OF THE DISSERTATION

# Autonomic Data Management for Extreme Scale Coupled Scientific Workflows

**By TONG JIN**

**Dissertation Director:**

**Manish Parashar**

Advanced coupled scientific simulation workflows running at extreme scales are providing new capabilities and new opportunities for high fidelity modeling and insights in a wide range of application areas. These workflows compose multiple physical models along with visualization and analysis services that share and exchange large amounts of data at runtime. Due to the huge I/O overhead, traditional file-based coupling approaches become infeasible. Instead, recent simulation-time data management approaches using in-memory data-staging methods have been explored to address this challenge. However, due to the complexities of emerging coupled applications and the architecture of current and future systems, these data staging based solutions are also presenting several new challenges.

First, many of these scientific workflows containing dynamically adaptive formulations, such as Adaptive Mesh Refinement (AMR), which exhibit dynamic runtime behaviors and result in dynamically changing data volumes and imbalanced data distributions. Such dynamic runtime behaviors increase the complexity of managing and processing simulation data. In addition, these behaviors introduce new challenges of managing the staging resources as well as scheduling in-memory data processing while satisfying constraints on (1) the amount of data movement, (2) the overhead on the simulation, and/or (3) the quality of the simulations/analysis. Second, architectural trends indicate that emerging systems

will have increasing numbers of cores per node and correspondingly decreasing amounts of DRAM memory per core as well as decreasing memory bandwidth. These trends can significantly impact the effectiveness of the online data management approaches for runtime data processing pipelines, and especially their ability to support data intensive simulation workflows.

To address the above dynamic data management challenges, this thesis explores an autonomic approach to enable efficient runtime data management, which can dynamically respond to the varying data management requirements. Specifically, it first formulates an abstraction that can be used to realize autonomic data management runtimes for coupled simulation workflows. To address the dynamic data management challenges in tightly coupled simulation workflows containing dynamically adaptive formulations, this thesis then presents a realization of this autonomic approach that uses runtime cross-layer adaptations. This realization explores autonomic runtime adaptations at application layer, middleware layer, and resource layer. It also exploits a coordinated approach that dynamically combines these adaptations in a cross-layer manner. This thesis also presents an autonomic multi-tiered data management runtime that leverages both DRAM and SSD to support autonomic data management for loosely coupled scientific workflows. It demonstrates how an autonomic data placement mechanism can dynamically manage and optimize data placement across the DRAM and SSD storage levels in this multi-tiered runtime realization. The research concepts and approaches have been prototyped and experimentally evaluated using real application workflows on current high end computing systems, including the Intrepid IBM BlueGene/P system at Argonne National Laboratory and the Titan Cray-XK7 system at Oak Ridge National Laboratory.

# Acknowledgments

First and foremost, I would like to express my gratitude to my advisor, Dr. Manish Parashar, for his guidance and support throughout my Ph.D study and research. I am truly grateful for his advice, encouragement, patience and cheerfulness.

I would like to thank to Dr. Ivan Rodero, Dr. Deborah Silver, and Dr. Hongfeng Yu, for serving as my Ph.D committee members and spending their precious time in reading and reviewing my thesis.

I would like to give many thanks to all the members of DataSpaces team, Fan Zhang, Hoang Bui, Qian Sun, and Melissa Romanus, for their generous help on my research and kindly encouragement during my Ph.D study. It is so wonderful to meet these amazing people and become a member in such an amazing team.

In addition, I wish to thank all the collaborators from different labs and universities for their helps, collaborations and co-authoring research papers. They include Dr. Scott Klasky, Dr. Norbert Podhorszki, Dr. Jong Y Choi, and Dr. Hasan Abbasi from Oak Ridge National Lab; Dr. Hongfeng Yu from University of Nebraska, Lincoin; Dr. Hemanth Kolla and Dr. Jacqueline Chen from Sandia National Lab; and Dr. Robert Hager and Dr. CS Chang from Princeton Plasma Physics Lab.

I would also like to thank all my friends and colleagues at Rutgers Discovery Informatics Institute (RDI2) for their kindly support and help during my Ph.D study.

My deepest gratitude goes to my parents Weihua Jin and Xiaojing Chen, and other family members, for their understanding, support, continuous inspiration and eternal love.

# Dedication

*To Veronica.*

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

### 1.1.1 Coupled Scientific Workflows

Advanced scientific 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 analytics and visualization services. For example, in the Community Earth System Model (CESM) [19] application workflow, separate simulations are coupled as part of a multi-physics workflow to model the interaction of the earth's ocean, atmosphere, land surface and sea ice. Another example is the turbulent combustion workflow, where a direct numerical simulation (DNS)[13] solver is coupled with multiple analytics components, e.g., descriptive statistics, visualization, and topology analysis. These coupled codes interact and exchange data at runtime to achieve high fidelity modeling for scientific discovery.

As system scale and application complexity grow, these workflows are exchanging very large amounts of data that must be processed and managed. The traditional file-based coupling approach, which supports data sharing between coupled applications within workflows using files, becomes infeasible and cost prohibitive from both performance and energy perspectives. Instead, recent runtime data management approaches[30, 5, 92, 49, 95, 25] using in-memory data-staging methods have been explored to efficiently support coupled simulation workflows, which have the potential to accelerate the overall time-to-solution process. In such an approach, the shared data will be stored using distributed in memory buffer across different nodes, called a staging area. Furthermore, based on how and where they are performed, the coupled applications can be deployed either *in-situ* or *in-transit*.

Specifically, the coupled applications are performed *in-situ* on the same compute resource where the primary simulation is running. In contrast, they can also be performed *in-transit* where part or all of the data is offloaded to a dedicated group of nodes on the same system for concurrent processing. For instance, in combustion simulation-analysis workflow[8], the analysis codes, i.e., visualization, descriptive statistic, and topology analysis, extract the output data at runtime and process them concurrently and in parallel while the primary simulation is running.

### 1.1.2 Dynamic Runtime Data Management Requirements and Challenges

With the increasing complexities of coupled applications and the architecture development of the High End Computing (HEC) systems, these coupled scientific simulation workflows running on extreme-scale computing system are presenting several new challenges.

First, many of the scientific simulations are based on dynamically adaptive formulations, which exhibit dynamic runtime behaviors and result in dynamically changing data volumes and imbalanced data distributions. For example, in AMR(Adaptive Mesh Refinement) based simulations, dynamic refinements can lead to imbalanced data distribution and heterogeneous resource (memory, CPU, network bandwidth) requirements. Such dynamic runtime behaviors increase the complexity of managing and processing the simulation data onlie while satisfying the constraints on the amount of data movement and the overhead on the simulation. They also impact on the management of the staging resources and the schedule of in-situ and/or in-transit data processing under the level of analytics and resource limitations. While current in-memory data management approaches can pre-configure the simulation in relatively simple static workflows, they becomes ineffective when application behaviors and data requirements become dynamic. Therefore, the effectiveness of runtime data management approaches for dynamic workflow depends on the efficient and dynamic mapping of workflow components, the size and distribution of the data and the resources available in-situ and in-transit, and requires adaptive configuring the staging resources based on dynamic application constraints and requirements.

Second, architectural trends indicate that emerging systems will have increasing numbers of cores per node and correspondingly decreasing amounts of DRAM memory per core as

well as decreasing memory bandwidth. These trends can significantly impact the effectiveness of the data management approaches in the runtime data processing pipeline, especially their ability to support data intensive simulation workflows and/or loosely coupled workflows that demand sufficient capacity for temporally caching a large volume of data. For example, in a data-intensive coupled simulation workflow[14], data generated by one simulation may have to be consumed by another simulation or by an analytics component before it can be overwritten. As data volumes increase and memory capacity and bandwidth decrease, memory constraints can quickly become a bottleneck of performing in-situ/in-transit processing. Furthermore, the coupled simulation and/or the analytics components may require the accumulation of data over multiple time steps[65], which can further accentuate this problem. Fortunately, hardware development like the usage of non-volatile memory devices, e.g. non-volatile memory (NVRAM) and solid state disks (SSDs), are becoming more pervasive and have been deployed on a number of systems, such as Gordon at Sand Diego Supercomputing Center (SDSC), Tsubame2 at Tokyo Institute of Technology, and Sith at Oak Ridge National Laboratory, which provides the potential to help with this issue. These new memory layers in extreme scale computers will provide a novel manner to share information and data across different nodes and application components, as well as maintain data persistence during simulation time. However, effectively using it may require runtime support for applications to be explicitly aware of these memory hierarchy layers and adaptively manage it in competition and coordination with other coupled components. Besides, additional complexities associated with managing the placement of data in different memory hierarchy levels and the data access by multiple concurrently executing tasks still present significant challenges.

Clearly, to address the challenges mentioned above, making staging-based data management approaches effective on current high end computing systems with new hardware technologies for these dynamic applications given performance, overhead and resource constraints requires autonomic approach and dedicated runtime support for runtime adaptations, trade-offs, and capability of self-controlling.

## 1.2 Problem Description

The dynamic characteristics of these applications and the trends in system architecture bring in new complexities and challenges on data management. There are four primary requirements and relevant challenges in the context of performing data management in extreme scale coupled scientific workflows.

**Computation scheduling and placement**: To improve the overall end-to-end performance and reduce in-network data movement, the coupled components have to be scheduled and placed appropriately during the runtime. Specifically, when scheduling and placing computations, several factors have to be considered, the sharing of processors, memory availability, the data exchange interaction between coupled components, as well as the cost of data movement. For example, AMR-based simulations involving local refinements can dynamically increase the resources consumed by the simulation on a subset of nodes, which in turn reduces the resources available for in-situ data processing. Meanwhile, the features of coupled components, such as their scalability, may also impact scheduling and placement. For example, some visualization and analysis routines are fundamentally memory-intensive and others are intrinsically compute-intensive, which are not suitable for in-situ processing. Furthermore, some applications, such as descriptive statistic analysis, can be performed in-situ, while others, such as topology analysis that needs heavy communication, should be performed in-transit.

**Data placement**: One of the key data management challenges and requirements is effectively and adaptively managing the runtime placement of simulation data. With the development of contemporary high performance systems and runtime data processing strategies, simulation data can be placed at multiple locations within the system, such as DRAM in primary computation resource, dedicated staging area, and/or levels of local deep memory hierarchy. Generally, the output data is supposed to be placed closer to the consuming components, e.g., analysis/visualization, which enables quick data access with lower overhead. In addition, it is important to assess the necessity and importance of data movement following some specific rules, such as potential data access patterns, in order to avoid redundant data movement horizontally through network and/or vertically across different levels

of local memory hierarchy. Finally, data placement should also take into account the resource constraints (e.g. DRAM availability) at the location where the data will be placed and managed. For example, in case of a loosely coupling workflow containing memory-intensive simulations, it is better to move output data out of the DRAM at local primary computation resource quickly, so that it does not run out of memory resource.

**Data quality management**: In coupled multi-physics multi-scale simulation workflows, such as the DNS-LES workflow, each simulation code could have its own discretization of space and time, and generate or consume data at different levels of spatial and temporal resolution. Similarly, in simulation-time data analysis workflows, the simulation generates data at a finer-grained level, while the analysis code may only need data at coarser resolution. This mismatch of data quality levels between data producer and data consumer can result in extra overhead due to both necessary data pre-processing and the movement of redundant data across network for performing in-transit analytics. As a result, it is required to adjust and manage data resolution dynamically, in order to meet the data quality requirement of coupled applications before data is moved. Additionally, the management of data quality should also be performed in an adaptive manner, taking into consideration resource constraints as well as evaluating appropriate tradeoffs such as performance and overhead.

**Utilization of staging resources**: Coupled simulation workflow involving dynamic behaviors will generate data in varying sizes and distributions at runtime, impacting the amount of resource required to perform data processing and analysis at runtime. For example, dynamic workflow increases the spatial and temporal resolutions of data, which in turn increases the computational and storage requirements of staging resources. These varying computational and storage requirements increase the complexity of configuring staging resources and impact the resource utilization efficiency. As a result, existing static resource pre-allocation strategies are not sufficient in such scenarios. In order to meet the needs of these dynamic workflows, staging resources(including both computation and storage) must have the ability to be dynamically allocated and utilized during runtime. This additional functionality enable the in-staging data analysis and improve the overall effectiveness of resource utilization.

## 1.3 Overview of Thesis Research

The overall goal of this thesis is to address the above dynamic data management challenges that scientists face in creating and managing simulation-time workflows in order to expedite insights into critical scientific processes at extreme scales. This thesis presents an autonomic approach to enable efficient data management at simulation-time, which can dynamically respond to the varying data management requirements at runtime. Specifically, this autonomic data management approach can efficiently enable the application and the runtime system to adapt their own behaviors – e.g., data quality adjustment, data placement, analytic placement, and resource allocation – automatically by following predefined strategies. These strategies are selected adaptively based on the varying operating environment statuses and specific objectives.

This thesis first presents a high-level abstraction of the autonomic approach for coupled scientific workflows running on extreme scale high end systems. Using this abstraction, specific characteristics and requirements for enabling such an autonomic approach can be identified. Specifically, it first describes the conceptual model of this approach, which contains three key elements: *autonomic objective*, *autonomic policy* and *autonomic mechanism*, which interact and coordinate with each other to achieve the overall autonomic data management goals. It then discusses how this autonomic approach can be realized on current high performance systems with scientific workflows. The discussion of such possible realization includes its runtime architecture, basic components, as well as the required capabilities and functionalities of such an approach. Finally, this thesis presents the realizations of this autonomic approach in two coupling cases and how it is used to effectively address different data management requirements and challenges.

To address the dynamic data management challenges in tightly coupled simulation workflows containing dynamically adaptive formulations, this thesis presents a realization of this autonomic data management approach by using runtime cross-layer adaptations. Specifically, this section of the work explores runtime adaptation policies and mechanisms implemented as a part of the autonomic runtime at three different layers, viz., application layer, middleware layer, and resource layer. The autonomic runtime is able to respond to dynamic data processing requirements and resource constraints for coupled AMR-based simulation

workflows. At the application layer, it dynamically adapts the spatial and temporal resolution of the data being written and processed under DRAM availability constraints and/or application requirements; at the middleware layer, it adapts the in-situ and/or in-transit placement and scheduling of data processing operations; and at the resource layer, it adapts the allocation of in-transit resources. It also presents 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. This runtime realization has been deployed and evaluated with real world applications and shown to be effective in improving overall performance and reducing unnecessary data movement.

This thesis also explores how the inclusion of a deep memory hierarchy, e.g., one involving solid state devices, can be used for data staging to adaptively deal with data management dynamics in loosely coupled workflows at extreme scales. Specifically, it presents an autonomic multi-tiered data staging runtime that leverages both DRAM and SSD to support autonomic data staging for coupled data intensive simulation workflows. By utilizing SSD, the presented hybrid staging approach can successfully accommodate larger volumes of data that may exceed the available DRAM main memory within the staging area, thereby supporting runtime code coupling and data management in a wider range of workflows. However, the inclusion of SSDs into the autonomic runtime introduces the additional levels of complexities. In order to overcome these challenges, this thesis also presents an adaptive application-aware data placement mechanism that can dynamically and automatically manage and optimize data placement across DRAM and SSD storage layers using data access patterns and user provided hints. More specifically, applications can specify and update spatial and temporal attributes describing their expected data access patterns as hints along with the workflow. At the same time, data access patterns are tracked at runtime and are used to predict future data access patterns. Furthermore, the runtime takes advantage of both the user's hints and the runtime prediction to assess the access probabilities and importance of each data object, and then quantifies these data access possibilities and importance by calculating and assigning a *utility value*. Data objects with higher utility value will be prefetched from a lower level of the memory hierarchy (i.e.,

SSD) to a higher one (i.e., local DRAM), and/or stay at higher memory hierarchy level longer, which is demonstrated to reduce the concurrent data access overheads and improves the overall time to solution.

## 1.4 Contributions

The primary contributions of the research in this thesis are insights into a systematic approach to support autonomic data management for extreme scale coupled scientific workflows. These insights will significantly enhance the capabilities and efficiency of scientific discovery for a variety of application domains. The detailed contributions are presented as follows.

- Systematic design and formulation of an autonomic data management abstraction for coupled scientific workflows in extreme scale High End Computing (HEC) infrastructure based on dynamic runtime data management requirements, characteristics of application behaviors, and environment of the system/resource. This provides the foundation for the conceptual framework of the approach and allows for the specification of its key components and required functionalities. This framework provides a variety of mechanisms that will be automatically orchestrated at runtime to respond to the heterogeneity and dynamics of the applications and the infrastructure.

- Design and development of effective cross-layer adaptation policies and mechanisms to enable dynamic data management in *tightly coupled* workflows containing applications exhibiting dynamic behaviors. These adaptations include (1) adaptation of the spatial resolution at which the data is processed, (2) dynamic placement and scheduling of data processing kernels, (3) dynamic allocation of in-transit resources, and (4) coordinated adaptations that dynamically combine these adaptations at the different layers automatically at runtime. This dynamic data management scheme reduces the in-network data movement, improves overall time-to-solution and increases resource efficiency at runtime.

- Design and development of a multi-tiered staging runtime that spans both DRAM and a deep memory hierarchy layer – solid state disks (SSD) – to support both dynamic

data management and code coupling for *loosely coupled* data intensive simulation workflows. This hybrid staging runtime allows entire workflows to cache larger data volumes that may exceed available DRAM memory within the staging area for runtime data management. In addition, this staging method can be deployed in future High End Computing systems by replacing the SSD layer with another faster memory hierarchy device, such as NVRAM.

- Design and implementation of an adaptive application-aware data placement mechanism that can dynamically manage and optimize data placement across the DRAM and SSD storage layers using data access patterns and user provided hints. This mechanism can support effective data prefetching from a lower level of memory hierarchy (i.e, SSD) to local DRAM in hybrid staging areas, reducing access overheads and improving the end-to-end performance.

- Integration of the prototype of the autonomic data management approach with real-word coupled scientific simulation workflows, e.g., combustion and fusion applications; and demonstrate its effectiveness in improving overall performance and accelerating scientific discovery.

## 1.5 Outline

The rest of this thesis is organized as follows.

Chapter 2 presents driving applications in real-world coupled scientific workflows that motivate the research work of this thesis. This chapter also summarizes the key parameters and components of coupled scientific workflows at exa-scale, as well as the data attributes (e.g. data resolution, data placement, etc.) that significantly impact the data management requirements in most of these workflows.

Chapter 3 presents an overview of related research work.

Chapter 4 presents an overview of the autonomic data management approach and its high-level abstraction, which contains the key elements and functions to support dynamic and adaptive responses at runtime to different data management and processing requirements. Moreover, it describes the execution model and the conceptual architecture of such

an approach, and characterizes the basic components needed for realizing it in different coupled simulation workflows at extreme scales.

Chapter 5 presents the design, implementation, and evaluation of a concrete realization of the autonomic approach for tightly coupled AMR-based simulation workflows at extreme scales, which utilizes cross-layer adaptations to enable dynamic in-memory data management. Specifically, it presents the detailed runtime adaptations at three different layers, viz., application layer, middleware layer, and resource layer, as well as a coordinated method that dynamically combines these adaptations at the different layers.

Chapter 6 presents the design, implementation, and evaluation of the other realization of such autonomic data management approach – a multi-tiered data staging that leverages both DRAM and SSD (solid state device) to support efficient and dynamic data processing in loosely coupled data intensive workflows. This chapter also presents an adaptive application-aware data placement mechanism that dynamically manages and optimizes data placement across the layers of the distributed memory hierarchy, as well as coordinates data movement and data sharing between the components of the application workflow so as to maximize its utility to the application and reduce access costs.

Chapter 7 summarizes the research work of this thesis and presents future research directions.

# Chapter 2

# Driving Applications and Requirements

## 2.1 Motivating Applications

The research of this thesis is driven by the dynamic data management and processing requirements of coupled data-intensive scientific workflows running at extreme scales. As the scale and complexity of scientific workflows have grown, so has the amount of data that they generate. Therefore, efficient data management is of paramount importance to scientists running on high-end systems. In order to show the impact of this research, real-world scientific applications with specific data coupling and data management requirements are illustrated below. The data management requirements of these workflows make them difficult to be realized with traditional data transfer and handling approaches. As a result, such applications, spanning a variety of scientific communities, serve as the driving use cases. Additionally, this chapter describes the abstraction of coupled simulation workflows and summarizes the key parameters that impacts their runtime data management requirements.

### 2.1.1 AMR-based Polytropic Gas Simulation-Visualization Workflow

#### Workflow Description

*3D AMR Polytropic Gas* simulation is an Adaptive Mesh Refinement (AMR) based simulation that simulates the behaviors of polytropic gas in a three-dimensional computational domain. This compute-intensive application implements the Godunov unsplit algorithm[1] for integrating systems of conservation laws (e.g., the Euler equations of gas dynamics), which requires a lot of local memory. In addition, this AMR-based simulation involves dynamic local grid refinements at runtime. As the simulation evolves, the data grid can be refined or coarsened, which may result in varying spatial-temporal data resolutions. The

dynamic changing data resolution will significantly impact the total size of output data and result in imbalanced resources (e.g., DRAM and CPU) consumption across different simulation nodes.

Moreover, a visualization code that allows scientists to visually monitor the simulation results (e.g., the density probability distribution and the variation of gases) is coupled with the AMR Polytropic Gas simulation. In such a workflow, the AMR-based simulation and the visualization service are tightly coupled on the fly, exchanging data at every $n$th ($n >= 1$) time step from simulation to visualization.

**Data Management Requirements**

The dynamic runtime behaviors of AMR-based simulation workflows increase the complexities of managing and processing simulation data. In addition, these behaviors bring in challenges of managing the staging resources as well as scheduling in-situ and/or in-transit data processing while satisfying constraints on (1) the amount of data movement, (2) the overhead on the simulation, and/or (3) the level of visualization. First, the increasing data size and imbalanced data distribution caused by domain refinement/coarsening may impact the execution of in-situ data visualization due to local DRAM resource unavailability. To enable the continuity of the execution of the entire workflow under local storage resource constraints, the workflow requires intelligent and effective data temporal-spatial resolution adjustment in order to achieve a trade-off between data quality and limited DRAM resource constraints. For example, the data resolution can be reduced under the tolerance of fidelity loss before the visualization service is performed, so that the visualization could continue executing without running out of memory. The data resolution adjustment must not impact the data resolution desired for information discovery and the level of analysis. Second, the varying data size and distribution may also affect the placement and scheduling of the coupled visualization service. For example, performing the visualization service in-situ may not always work in cases where the local DRAM cannot accommodate both the increasing simulation data after refinement as well as the memory requirement for the visualization algorithm. On the other hand, performing the visualization service in-transit introduces high amounts of data movement across network. Therefore, it requires to place and schedule the

analytics flexibly and dynamically on the fly. Lastly, after the computational grid is refined in the AMR simulation and the visualization service has to be performed in-transit, the workflow will experience a dramatic increase in data volume which results in a demand for more DRAM availability and computing capabilities in staging area. Clearly, the existing approaches that preallocate staging resources at the beginning of the workflow will impact the overall utilization efficiency of in-transit computing resources. As a result, this workflow requires dynamic system configurations and adaptive allocations of in-transit computational and storage resources.

### 2.1.2 End-to-End Combustion Simulation-Analytics Workflow

#### Workflow Description

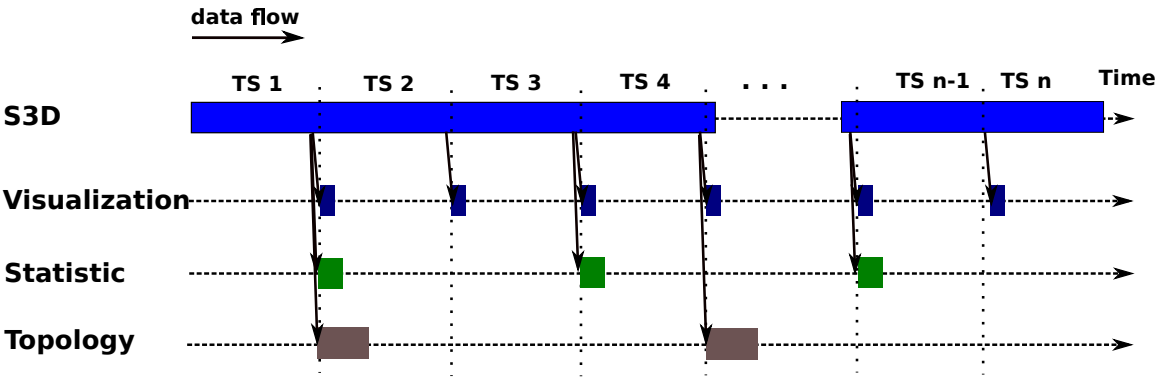



Figure 2.1: Illustration of data exchange and sharing flow and coupling relationship between coupled applications in the combustion S3D simulation-analytics workflow. Three analytics applications: visualization, descriptive statistic analysis, and topology analysis are coupled with the S3D simulation and consume output data following different temporal-spatial access patterns.

Massively parallel turbulent combustion simulations running at extreme scales introduce spatial and temporal scales spanning typically at least 5 decades. A popular scientific combustion workflow exhibiting this behavior is S3D [13], which performs first principles-based direct numerical simulations of turbulent combustion in which both turbulence and chemical kinetics associated with burning gas-phase hydrocarbon fuels. This kind of simulations can generate large amounts of data. For example, a simulation of flame stabilization by auto-ignition in a lifted hydrogen jet flame [89] produces more than 100GB data in approximately 17 seconds or less.

Statistically stationary and temporally evolving flows are two important categories of turbulent combustion problems simulated with S3D that require analytics to be performed at a high frequency. Traditional file-based data analytics manner that accesses data every few hundred time steps is cost-prohibitive and infeasible due to the fact that it cannot meet high frequency data analysis requirements. Therefore, in this workflow, multiple analytics codes, e.g., visualization, descriptive analysis, topology analysis, and etc., are coupled with the S3D simulation, performing different temporal and spatial data access patterns[8] and execution requirements.

Figure 2.1 illustrates the data exchange and sharing flow and coupling relationship between these applications in a combustion simulation-analytics workflow example. As shown in this figure, these analytics have varying degrees of coupling tightness respectively with the simulation, and perform different temporal and spatial data access patterns. For instance, topology analysis may access data less frequently than a descriptive statistic code does, while they require data from the same region of the entire data domain.

#### Data Management Requirements

In this workflow, simulation output data may be consumed concurrently by multiple coupled analysis code following different temporal and spatial data access patterns. As a result, the output data must be cached in the workflow and persist during runtime until it is consumed. Additionally, to preserve the spatial-temporal fidelity behind the data, the workflow requires frequent and timely exchange of chemical species data from the S3D simulation to the analysis services. This introduces overhead for the simulation that ultimately impacts the overall time to solution of the workflow. Therefore, in order to overcome these challenges and accelerate scientific discovery, the desired data must be effectively prepared and adaptively placed closer to each application ahead of time.

### 2.1.3 Coupled Combustion Simulation DNS-LES Workflow

#### Workflow Description

S3D[13] is a massively parallel computational fluid dynamics (CFD) solver that performs first principles based “direct numerical simulations” (DNS) of turbulent combustion. DNS

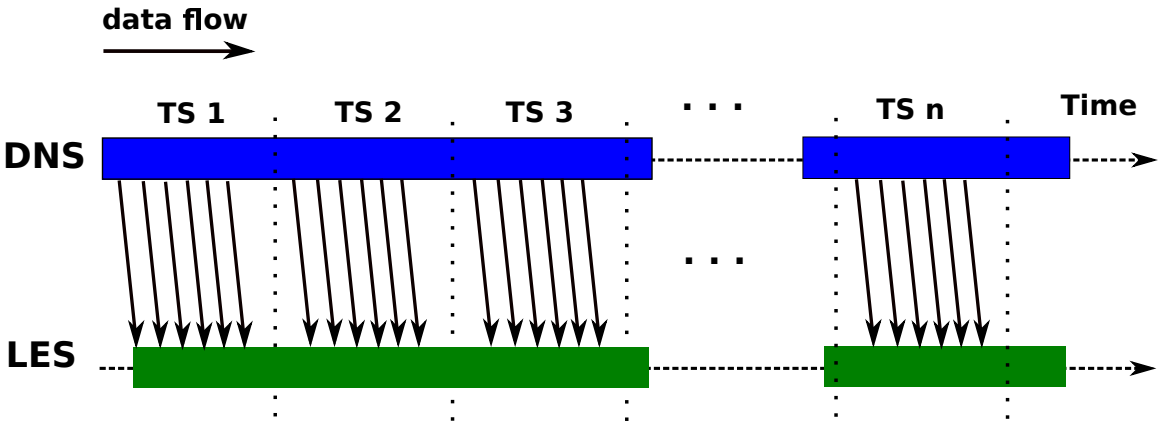



Figure 2.2: Illustration of normal data exchange and interactions between coupled DNS-LES simulation workflow. DSN and LES are tightly coupled together, performing concurrently and sharing data 6 times per time steps in this example.

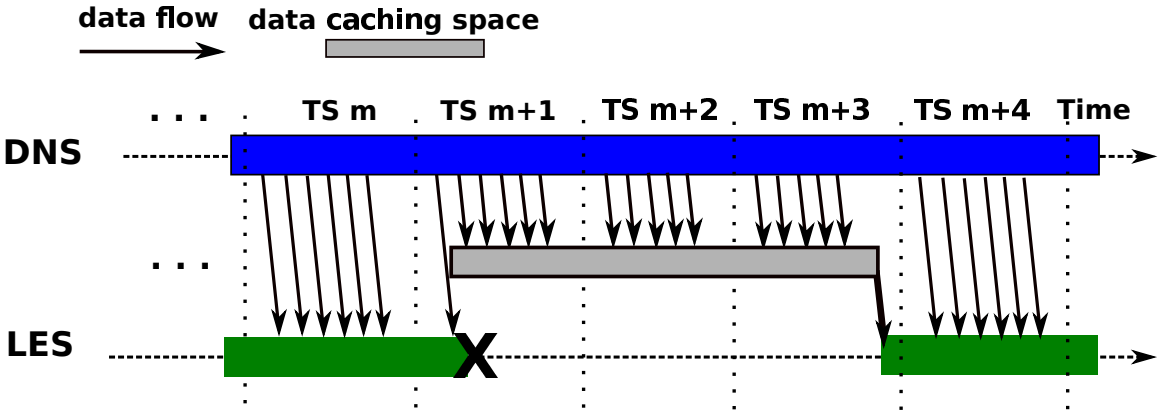



Figure 2.3: Illustration of an example where DNS and LES simulations go out of sync because of the lag of LES simulation, and then get recovered after 3 time steps. The data generated during these three time steps needs to be cached on the fly before the lock-step of data exchange recovers.

is very expensive both in terms of FLOPS and data generation, since it resolves the entire range of spatial and temporal scales in the continuum regime of a given problem. Another paradigm of turbulent combustion simulations that is less expensive and more suitable for engineering calculations is "large eddy simulations" (LES)[75], which only resolve the large energy containing a range of scales and models the physics for smaller scales. Coupled DNS-LES simulations are being considered as a rigorous, albeit expensive, test bed for assessing models, by isolating and eliminating numerical errors. This is achieved by performing DNS and LES in lockstep, where the base solution field from the well resolved DNS solution is appropriately filtered, and then fed to the LES simulation, which is running in tandem and is solving only one additional quantity of the model that is being tested. The grid sizes and time steps in both the DNS and LES are kept identical to eliminate numerical errors. Figure 2.2 illustrates an example of normal DNS-LES workflow and the flow of data exchange between these two simulations. In this instance, DNS and LES execute concurrently with internal one-directional data exchange to form a locked pipeline. Data is transferred from DNS to LES several times per time step (in this figure, this value is specifically 6 times per time step); and each of these exchanges costs a few seconds.

**Data Management Requirements**

The lock-step DNS-LES coupling requirement presents a number of data management requirements and challenges. First, the simulations advance in time using a six-stage Runge-Kutta scheme, which implies that the exchange of data between DNS and LES must happen six times at every time step, with each time step typically taking a few seconds of wall-clock time. This pattern leads to frequent exchanges of a large volume of data. Second, due to several uncertainties during execution (e.g., lag of LES simulation, network issues while transferring the data), the DNS and LES simulations often go out of sync for so many time steps (as demonstrated in Figure 2.3). As a result, the exchanged data needs to be cached during runtime over multiple time steps into a large capacity storage space, before the lock-step data exchange can recover. However, DRAM-only staging approaches cannot accommodate the large data sizes, because this data easily fills up the available DRAM storage capacity.

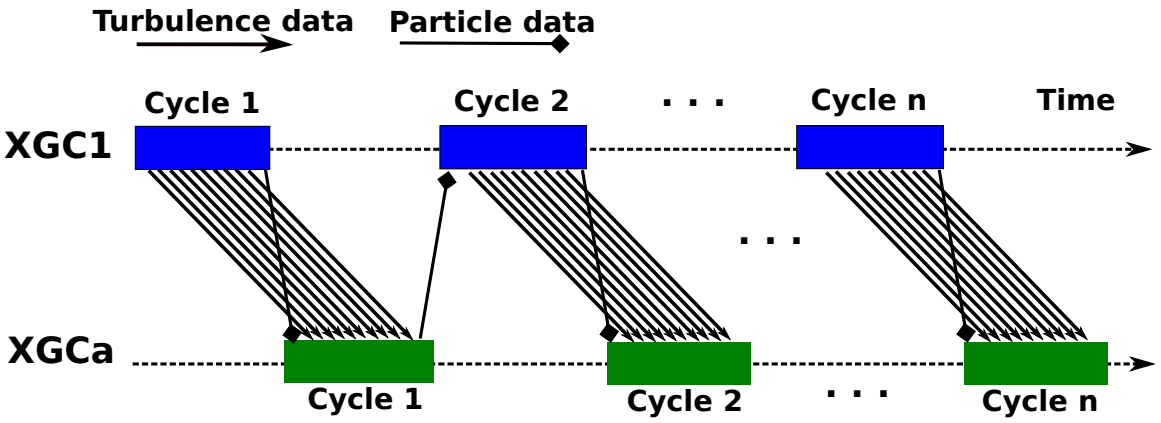



Figure 2.4: Illustration of data exchange and interactions between coupled plasma fusion XGC1-XGCa workflow. Two types of data - turbulence data and particle data - are exchanged between XGC1 and XGCa.

### 2.1.4 Coupled Plasma Fusion Simulation XGC1-XGCa Workflow

#### Workflow Description

XGC1 [44] and XGCa are extreme-scale, first-principles particle-in-cell (PIC) codes for the simulation of magnetically confined plasmas in nuclear fusion research. Due to numerical dissipation, model equation errors and the huge computational costs of XGC1 simulations, XGCa, which uses a coarse mesh for the Poisson solver and requires much fewer particles, is coupled with XGC1 as a complimentary accelerator.

The execution of XGC1-XGCa workflow consists of multiple coupling cycles, as illustrated in Figure 2.4. In each coupling cycle, the workflow first executes XGC1 for up to $n$ time steps ($n$ is 10 in this illustrated figure, but may be up to thousands in a regular case), generating the turbulence information at each time step and particle state data once. After the plasma has evolved to a quasi-steady turbulent state after an initial transient phase, execution is switched from XGC1 to XGCa. All the recorded turbulence information and the current state of the plasma are then read by XGCa in order to replay the pre-recorded turbulence information and advance the plasma background. When the background has evolved sufficiently to affect the turbulence, XGCa stops and returns the updated particle data to XGC1. Then, the next cycle is executed.

**Data Management Requirements**

When the workflow passes control between XGC1 and XGCa, it needs to transfer data containing velocity and positions information of $10^{10}$ particles, the size of which is approximately 85 bytes per particle. Based on scientists' experience, the total amount of particle data is about 4 Tera bytes for a typical XGC1 simulation of the DIII-D tokamak [59] running on 16,384 compute nodes on Titan, with 1.6 million particles per compute node and 2 particle species (ions and electrons). Furthermore, for the much larger tokamak ITER, it may be 5 to 10 times or more because more particles are necessary to keep the sampling noise low. Therefore, the workflow requires larger storage space to cache this large portion of data on the fly, since it can easily exceed the storage capacity of a DRAM-only in-memory staging area. Meanwhile, locality-aware data placement is also required and desirable so that the overhead of concurrently querying such large quantities of data can be minimized.

## 2.2 Abstraction and Key Parameters of Coupled Scientific Workflow

Figure 2.5 illustrates the abstraction of coupled scientific simulation workflows, in which the primary scientific simulation is the data producer and secondary simulations, analytics, and/or visualization applications serve as data consumers coupled to the producer. These components run at different scales, produce and consume data at different rates, and progress at different speeds. Moreover, they regularly perform data exchange at runtime with different data size, data exchange frequencies, and spatial and temporal data access patterns. *Attributes of coupled computations*, *data attributes*, and *coupling relationship* are three of most important factors that impact the coupled scientific workflows and characterize the relevant execution and data management requirements. The following paragraphs lists and summarize the key parameters of these three factors.

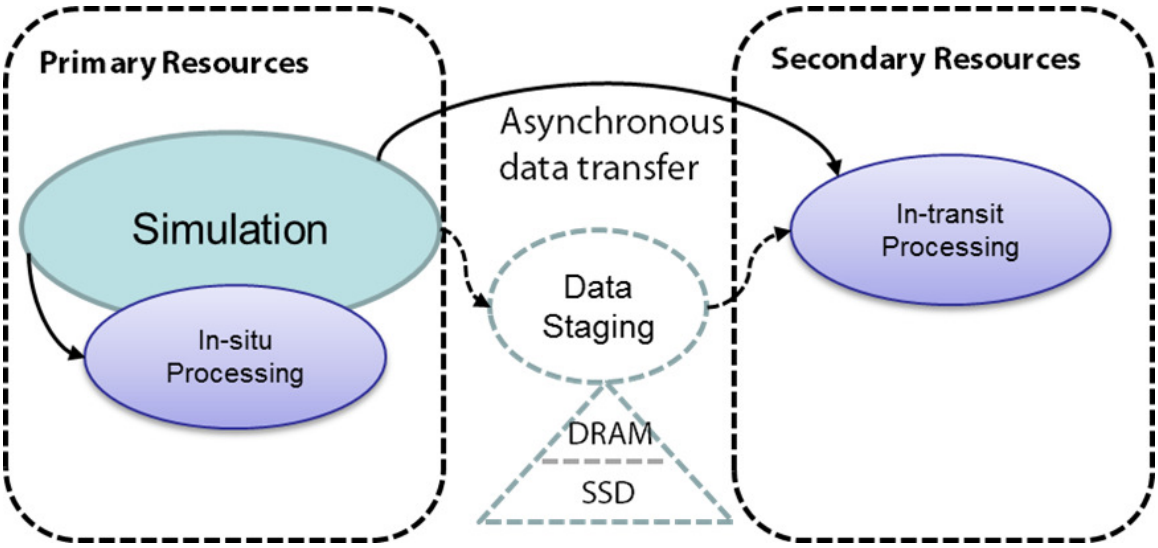



Figure 2.5: Overview of the coupled simulation-analytics workflow. "Primary Resources" refer to where the main simulation is running, while "secondary resources" refer to a dedicated group of computing nodes running coupled simulation or analysis services.

### 2.2.1 Attributes of Coupled Computations

**Locality of Coupled Computation**: The most important key attribute of coupled computations in a coupled simulation workflow is their locality, which determines the placement of computation components to physical processor cores across the distributed compute nodes. The placement of computation would influence the performance of the entire workflow execution particularly in terms of the amount of network data movement and cost of parallel communications, as well as the effectiveness of resource utilization. Generally, the coupled computation components can be performed *in-situ*, *in-transit*, or in a hybrid manner. The difference between them lies in how and where the computation is performed. *In-situ* analysis typically shares the primary simulation compute resources. In contrast, when analyses are performed *in-transit*, some or all of the data is transferred to different processors, either on the same machine or on different computing resources all together. Hybrid data processing combines both of these two methods, performing the scalable low-overhead part of analysis in-situ, and the rest in-transit.

### 2.2.2 Data Attributes

**Data Locality**: Locality of data significantly impacts the coupling behaviors and data management requirements in coupled scientific workflow at extreme scales. Different from the traditional concept of data locality in memory hierarchy, data locality in relation to a coupled simulation workflow generally refers to the places where the simulation output data is put temporarily or persistently for data processing or management. There are two types of locality of data: *vertical locality*, which defines the placement of data in different memory hierarchy levels such as DRAM, NVRAM, SSD, and even in disk, and *horizontal locality*, which refers to the placement of data on different computational resources in a coupled simulation workflow, e.g., using the primary computational resource for in-situ processing and/or the secondary computational resource for in-transit data management.

**Data resolution**: In scientific applications, the physical domain of a particular scientific problem will be mapped into a computational domain, which is a simplified form of the physical domain in terms of geometrical representation and boundary condition imposition. This simplified form should keep all the important physical features of the scientific domain problem by defining relevant data as variables in the computational domain. As the simulation is running, values of element data variables in the computational domain keep refreshing according to their physical behaviors. *Data resolution* refers to both the density of the elements in a specific computational domain and the precision of a measurement with respect to time, such as the frequency of data value refresh or the the frequency of data access. The former is called data *spatial resolution* and the later is data *temporal resolution*. The level of data resolution impacts on the size and distribution of the data to be managed, analyzed and visualized, as well as the potential insights that can be gleaned from the data. For example, in AMR-based simulations, the data regions maybe further refined or coarsened as the simulations evolve, which can result in imbalanced data distribution across parallel computing resources and corresponding resources constraints.

### 2.2.3 Coupling Relationship

**Coupling tightness**: Data interactions and computation coupling of a typical simulation workflow can be categorized as either *tight coupling* or *loose coupling*, which indicates the

ratio of the effort (e.g. wall-clock time) by inter-component data interactions versus intra-component computation. In *tightly coupled* workflows, the coupled components exchange data directly from component to component with high frequency (possibly at each time step), as in the case of the polytropic gas simulation. The data consumer normally consumes the data quickly on the fly. In *loosely coupled* workflows, the data is exchanged less frequently, often asynchronously and possibly opportunistically. An example of this pattern is the XGC1-XGCa [44] coupling in the end-to-end fusion simulation workflow. Usually, data aggregation and accumulation occur in this coupling scenario, requiring the temporary caching or buffering of data (e.g. in the staging area or local deep memory hierarchy devices) for later simulation-time usage.

**Coupling cycle and frequency**: Coupling cycle is the period over which all components of the coupled scientific workflow have exchanged data at least once. Coupling frequency determines the rate of data exchange for a given workflow. For example, in-staging computation, especially for well-endowed staging nodes, probably can operate on multiple output steps and will be bound by timing requirements determined by output frequencies. Thus, it directly impacts the data management requirements and complexities on the fly and consequently the strategy of the scheduling and placement of coupled analytics.

# Chapter 3

# Background and Related Work

## 3.1 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. Recently, a number of runtime data processing approaches have been proposed in order to accelerate scientific discovery by performing in-memory data analysis, These research projects have focused on three specific simulation-time analysis techniques, namely in-situ processing, in-transit processing, and hybrid processing.

**In-situ data processing**: In-situ data processing typically shares the primary simulation compute resources and allows direct access to in-memory simulation data. It has been used in visualization [60], [90], [30], indexing building [41], data compression [45], multi-physics coupling [92], etc. This technique greatly reduces the costs due to data movement across the network because most data is available in local memory.

There are two primary branches of *in-situ* techniques recently: *inline processing* approach and *helper cores* approach, both of which utilize the same resource of primary as simulations. *Inline processing* refers that analysis/visualization codes are synchronously executed with the simulation at runtime iteratively. There are many applications e.g. ParaView [30] and VisIt [11] visualization using such approach for in-situ processing and visualizations. On the other head, *helper cores* approach, such as CoDS [92], leverages some local CPU cores of nodes where simulation is running to perform dedicated analysis, processing, or visualization in parallel. Examples include Functional Partitioning [49] and so on.

However, due to the resource sharing nature of in-situ processing, it can increase the

overall time-to-solution and coupling complexities. Also, many data analysis algorithms are global in nature and few of them have capability of scaling satisfactorily.

**In-transit data processing**: In-transit data processing offloads raw data to dedicated compute resources, i.e., a set of additional compute nodes allocated by users when launching the parallel simulations, and executes it there 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 [5], PreDatA [95], DataSpaces [27, 28]/ActiveSpaces [25], GLEAN [81] and Nessie [66].

*PreDatA* middleware (preparatory Data Analytics) [95] augments the current I/O stack on HEC platform with data staging and processing by exploiting computational resources in both compute nodes and dedicated staging nodes. Key features of PreDatA include asynchronous data movement from compute to staging nodes, pluggable framework for performing user-defined data analysis operations. Similarly, *CLEAN* [81] is a data staging substrate that enables improved data movement between compute nodes and staging nodes, interfacing to running simulations for in-transit co-analysis or in-situ analysis. GLEAN exploits the network topology of IBM BlueGene/P system and proposes the design of a topology-aware data movement and aggregation mechanism to improve the performance of I/O forwarding. However, it does not specify user-level programming APIs to enable a programmable data staging substrate. Another typical example is *DataSpaces* [27, 28], which is a programming runtime targeted at current large scale systems and designed to support dynamic interaction and coordination patterns between scientific applications. It provides the programming APIs to enable flexible in-transit data processing, and utilizes RDMA to support asynchronous data transfer between simulation resources and staging resources. On the top of *DataSpaces*, *ActiveSpaces* [25] takes a step further to support dynamic deployment of user-defined data processing routines onto dedicated staging nodes. The runtime system distributes the serial data kernel to multiple staging nodes where the queried data resides and gathers results returned by each kernel execution.

Most of these existing data staging solutions primarily focus on fast and asynchronous data movement off simulation nodes to lessen the impact of expensive I/O operations. They

typically support limited data operations within the staging area, such as pre-processing, and transformations, often resulting in under-utilization of the staging nodes' compute power. Also, the data movement across the network in this approach can introduce large overheads as well as increase power consumption.

**Hybrid data processing**: To take advantage of both in-situ and in-transit data processing approaches, recent research [8] has explored the benefits of combining both in-situ and in-transit approaches on leadership class supercomputers, and demonstrated the importance of executing the analytics in a hybrid in-situ/in-transit staging system. It forms a multi-stage pipeline to support various simultaneous analyses by utilizing the data buffering and computation capabilities of both local nodes and staging nodes. Another instance of hybrid data processing is *FlexIO* [96], which explores the trade-offs between performing analytics at different levels of the I/O hierarchy and supports a variety of simulation-analytics workloads through flexible placement options. *FlexIO* enables users to tune the placement of analytics under the objective of overall performance of data analytics workflows, and then automatically configure the underlying transportation methods to support the computation placement decisions. *JITStager* [4] enables users to apply customized data operators to simulation output data along with the entire I/O pipeline. It implements SmartTap to execute data operations in-situ on processor cores that perform the simulation code. Moreover, Since *JITStager* is built on software DataStager [5], it can also extract data from simulation nodes to staging nodes, execute the data customization operators in staging, and then forward the data to downstream data analysis for further processing.

However, these research efforts target static application workflows and pre-schedule the placement of the analysis components.

## 3.2 Autonomic Technologies and Related Work

Kephart and Chess [40] have defined autonomic computing as a general computing approach where computing systems can manage themselves given specific pre-defined objectives. The ultimate goal of autonomic computing is to develop computing systems capable of self-management to deal with the dramatically growing management complexities. Many

research work have focused on this area with different objectives, such as improving application performance [53], optimizing Quality of Service [85], increasing resource utilization [46]. Moreover, many comprehensive surveys [69, 23, 15, 24] of autonomic computing have also been published.

**Self* property of autonomic computing**: The primary properties of a self-management autonomic system are defined by IBM [37], referred to as self-star (self-*) properties, which contain four properties: *self-configuration*, *self-optimization*, *self-healing* and *self-protection*. *Self-configuration* refers that an autonomic computing system configures itself according to high level goals. *Self-optimization* refers that an autonomic computing system optimizes its use of resources to ensure the optimal functions by following specific requirements. *Self-healing* refers that an autonomic system should be able to discover and correct itself from faults. *Self-protection* refers that an autonomic system can protect itself from malicious and arbitrary attacks.

**Adaptation plans**: Adaptation technologies can be integrated with runtime systems to enable autonomic self-managed applications. There are many different types of adaptations plans and strategies.

*Control-based adaptation* is derived from control theory and utilizes feedback control to achieve defined goals. Many research efforts [74, 57] have investigated using feedback or reactive control for resource and performance management. The techniques proposed in these research work attempted to take corrective actions based on the observations of current application states, in order to achieve specific objectives. Control-based adaptations have been successfully applied to several of domains, such as web service QoS adaptation [6], task scheduling [57], load balancing [56], and processor power management [74].

*Model-based adaptation* has been used in a variety of contexts by using specific adaptation models. Most of these adaptation strategies focus on the usage of specific models, e.g., performance models to support load balancing, other than general problems. For example, many researchers explore architectural models [67] for model-based adaptations, which utilize specific styles of architectures in their systems. There are many of such cases – Taylor's work used a hierarchical publish-subscribe model through C2 [68]; and Magee used bi-directional communication links via Darwin [63].

*Rule-based adaptation* performs all the adaptations following the pre-defined and generated rules that target at the final objectives. For example, ACT [73] uses a rule-based interceptor to adaptively weave new codes while executing applications. ALua [80] uses Lua language to perform adaptations in an interpretive manner. RESAS [9] supports rule-based adaptations for real-time software and provides programming tools to the users. In addition, this adaptation technology has also been applied in scientific applications workflows, such as Grid services [50] and I/O optimization [61]. Recent research work of a programming framework 'Accord' [51], which was designed for autonomic applications, also systematically studied the rule-based adaptations.

*Policy-based adaptation*, similar to *rule-based adaptation*, essentially utilizes pre-defined policies to determine the actions to take when a specific event occurs and/or certain conditions are met. These polices are designed and described inside the computing systems, specifying the adaptation plans of the systems. There are many examples of using such an adaptation plan in scientific workflows. For example, [18] designs the policies of workflow management for data intensive applications; and [7] attempts to use policy-based data placement to improve overall workflow performance. Besides, [16, 17, 31] are all other examples of research that rely on such an adaptation technology.

## 3.3 Single-Layer and Cross-Layer Adaptation

Previous research efforts have focused on improving performance by using a single-layer adaptive approach. Tapus et al. [76] 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. [36] 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. [72] 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 [91] 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 [48].

## 3.4 SSD in High Performance Computing

New technologies such as solid state storage (SSDs) and non-volatile memories (such as PCM and memristors) provide a hardware based solution to this I/O performance bottleneck. NAND flash-based SSDs have many advantages over magnetic hard disks. For example, SSDs use less power than hard disks and at the same time provide lower access latency. As a result, SSDs are being increasingly used in HPC storage systems at different levels and for a variety of purposes.

One direction of SSD utilization is to use them as an intermediate storage level between DRAM and hard disk for data checkpointing. Several studies such as [71, 49] have investigated using compute node-local SSDs as storage buffers, for example, to temporarily cache checkpoint data to support recovery from failures. Although these efforts reduce the I/O overhead by selecting the appropriate memory hierarchy level as the data buffer between DRAM and hard disk, their focus was not to improve or accelerate the data-to-insight discovery process.

Research efforts have also explored using SSDs for data analysis. For example Active Flash [78] proposes in-situ scientific data analysis by directly executing data analysis tasks on emerging storage devices. Minerva [21] extends the conventional SSD architecture using a FPGA-based storage controller to offload data or I/O intensive application code to the SSD to accelerate data analysis. However, these solutions do not target data coupling for coupled application workflows, where the data exchange and access patterns can be both complex and dynamic. Furthermore, some of these solutions require hardware modifications

or special access priorities to the HPC systems, which can limit their use to specific resources.

## 3.5 Data Placement and Access Pattern Prediction

Many research works have attempted to explore efficient data placement within scientific workflows using different methods. For example, Stock [42] provides grid-based data placement service and utilizes adaptive network protocols to improve data movement efficiency. Reorganizing data inside file system, e.g., in research work [88, 77], is another popular approach to optimize data placement and data access performance. In such an approach, little effect has been involved in the overall workflow communication since parallel file system itself has dedicated system storage and network resources for enabling the optimization. Moreover, scientific workflow management systems, such as Kepler [58], Pegasus [22], and DAGMan [64], also contain different data placement and management strategies by using user provided information, e.g., the sequence of execution and/or the order of data exchange. The data is stored in files and is moved amongst applications through workflow engines. However, these systems have a limited capacity of handling dynamic workflows. Several other research efforts have used replication to optimize placement for parallel/distributed storage systems, such as HDFS [10] and GPFS-SNC [34]. In these approaches, multiple copies of data files are placed at different locations; and then the systems [10, 34] would select the 'best' (in terms of access cost) replica for accesses.

Besides the above data placement strategies, specific data access patterns have also been commonly used in developing storage systems to improve I/O performance. For example, GFS [32] is optimized for large data sets and appends accesses; EDO [77] explores space filling curve (SFC) based data organization to improve the data read performance for scientific applications. PDLA [88] optimizes data layout based on I/O behaviors captured in traces, to improve the data access performance of parallel I/O systems. However, please note that the optimization techniques in these systems are designed for static access patterns, and may become less effective when applications have dynamically varying data access patterns.

Due to the potential benefits of using data access patterns, the strategies of predicting data access patterns have been widely investigated. Many of these work have been successfully applied to either prefetching, caching or scheduling. Basically, these methods can be

classified as three categories: (1) statistical model based methods, (2) symbol-based methods, and (3) execution-based methods. In examples of model based methods, Madhyastha and Reed [62] used hidden Markov models and artificial neural networks to classify access patterns so as to improve the performance of file systems. Tran and Reed [79] used ARIMA time series to model inter-arrival time between I/O requests to predict temporal data access patterns. In examples of symbol-based methods, the contexts/symbols are used as parameters in some spatial access pattern prediction models [43]. Also, LZ77 sliding window algorithm [35] had been proposed to perform pattern aware prefetching. Omnisc'IO [29] built a grammar-based model of the I/O behavior of any HPC applications and used it to predict the temporal feature of future I/O operations (i.e., inter-arrival time between two I/O requests). After that, the spatial attribute of data access will be assessed (i.e., the file being accessed as well as the starting point, offset and size of the data in file). In the third category of method, Zhang et al. [94] utilized an extra new process to pre-execute the code to get future access patterns, so that main process could leverage it as a spatial and temporal access prediction. Other research work such as [12, 33] are also included into this category. Please note here, many of these existing prediction approaches rely on offline model training or pattern detection; and only a few of them can predict both temporal and spatial future access patterns.

# Chapter 4

# Autonomic Data Management Abstraction

## 4.1 Overview

Addressing the dynamic data management requirements of extreme scale coupled scientific workflow requires fundamental changes are required in the way that the application workflows are executed at runtime. Specifically, an autonomic data management approach has to be investigated, which should be able to monitor the operating environment and running applications, and then adapt and tune the application behaviors and resource allocations at runtime while meeting the data management requirements and constraints.

The overarching goal of this research work is to explore an efficient autonomic runtime approach that can autonomically respond to runtime dynamic data management requirements and resources constraints for coupled simulation workflows at extreme scale. Specifically, this autonomic data management approach is able to support the applications and/or the runtime to adapt and modify their behaviors, e.g. data resolution, data placement, placement of analytic, or resource allocation, autonomically. This is achieved using pre-defined strategies/policies that are triggered by the changes in operating environment. Similar to the conventional concept of autonomic computing [69, 23, 15, 24], this research concentrates using self-management to deal with the rapidly growing complexities and costs of runtime data management at extreme scales.

This chapter gives a schematic abstraction of the autonomic data management approach for extreme-scale high performance computing. It first presents a conceptual model for runtime data management; and then, describes the autonomic data management components as well as the overall autonomic execution model. Finally, it discusses the realization of the autonomic approach on current High Performance Computing systems in the context of coupled scientific workflows.

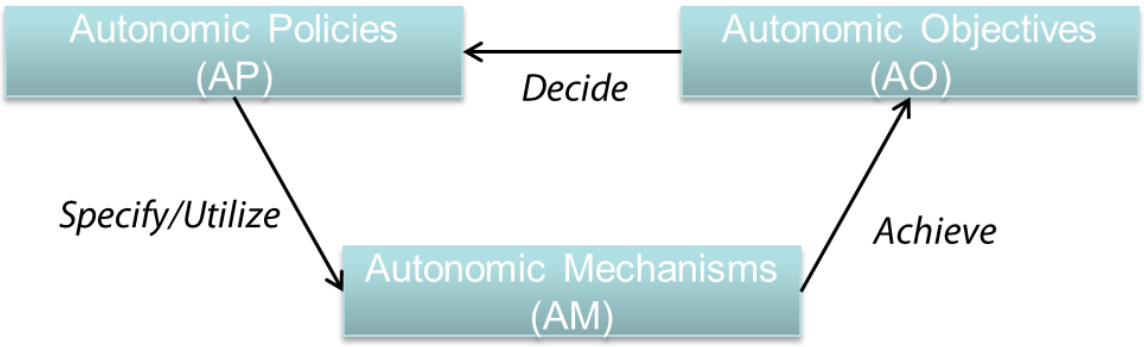



Figure 4.1: A conceptual model of the autonomic data management approach for coupled scientific workflows. Three key elements (AO, AP, and AM) and their associated relationships are illustrated.

## 4.2 Conceptual model of Autonomic Data Management

In this section, a conceptual model of the autonomic data management approach for coupled scientific workflows is presented and discussed. Conceptually, an autonomic data management framework should comprise of three key elements: *autonomic objective*, *autonomic policy* and *autonomic mechanism*. These three basic elements interact and coordinate with each other to achieve the autonomic adaptations. The elements and relationships among the elements are illustrated in Figure 4.1.

**Autonomic Objective (AO)**: An AO represents the goal that has to be achieved through adaptations. It corresponds with an application requirement that is typically defined by a user/scientist at the beginning of the job. Examples of AO can be "minimal end-to-end time", "least data movement", "most efficient usage of resources", "minimal power consumption", etc. Please note that multiple AOs may be specified simultaneously to be satisfied by following certain associative relationships.

**Autonomic Policy (AP)**: An autonomic policy (AP) refers to a particular pre-defined adaptation strategy or rule used to serve and accomplish user-defined AOs. For example, "data placement policy" defines the rules of placing data adaptively; "data movement policy" specifies the strategies of moving data across network; etc. One specific AP can be triggered in different scenarios for serving different AOs; and multiple APs can also be triggered in a combined manner to serve the same AO. As illustrated in Figure 4.1, Autonomic Policies are driven and organized by one or more particular AOs, and then specify

and utilize a set of specific autonomic mechanisms (AMs) to achieve the goals. Moreover, an AP can be configured either statically at the beginning of the job or autonomically at runtime by defining several of input arguments that are relevant to the application and the runtime environment. These inputs coming from the runtime operating environment and applications will then configure the APs to specify what autonomic mechanisms should be selected and how to execute them.

**Autonomic Mechanism (AM)**: An AM is a particular action that can be executed by the application or the runtime to achieve specified AOs. In coupled scientific workflow, an AM can be "performing in-transit analysis", "doing in-situ data down-sampling", "moving data", "adjusting CPU frequency", etc. As illustrated in Figure 4.1, when and how should the applications or the runtime systems execute these AMs are specified in autonomic policies (APs).

In principle, these basic elements define (1) why the adaptations are needed, (2) how these adaptations can be achieved, and (3) what actions should be exactly executed. To better demonstrate the importance of these elements, Figure 4.2 presents examples of autonomic elements relevant to a scientific computing environment, as well as their interaction and coordination relationships. Additionally, it also presents example input information sources, output controlling, and other interactive relationships between autonomic elements and the external computing environment. As shown in the figure, there are two types of input information sources, i.e., *pre-configured information* and *runtime status information*. *Pre-configured information* refers to user-provided information, such as domain information, application knowledge, etc., which can be defined through user interfaces or configuration files at the beginning of the job. And *runtime status information* refers to information relevant to operational state of systems and applications, e.g., resource availability (memory, CPU cores), application execution time, the size of generated data, etc.

## 4.3 Realizing Autonomic Adaptation in Coupled Scientific Workflow

To realize an autonomic data management runtime in the context of coupled scientific workflows on High End Computing systems, three important properties have to be achieved: *awareness*, *adaptive*, and *autonomic*. *Awareness* states that an autonomic data management

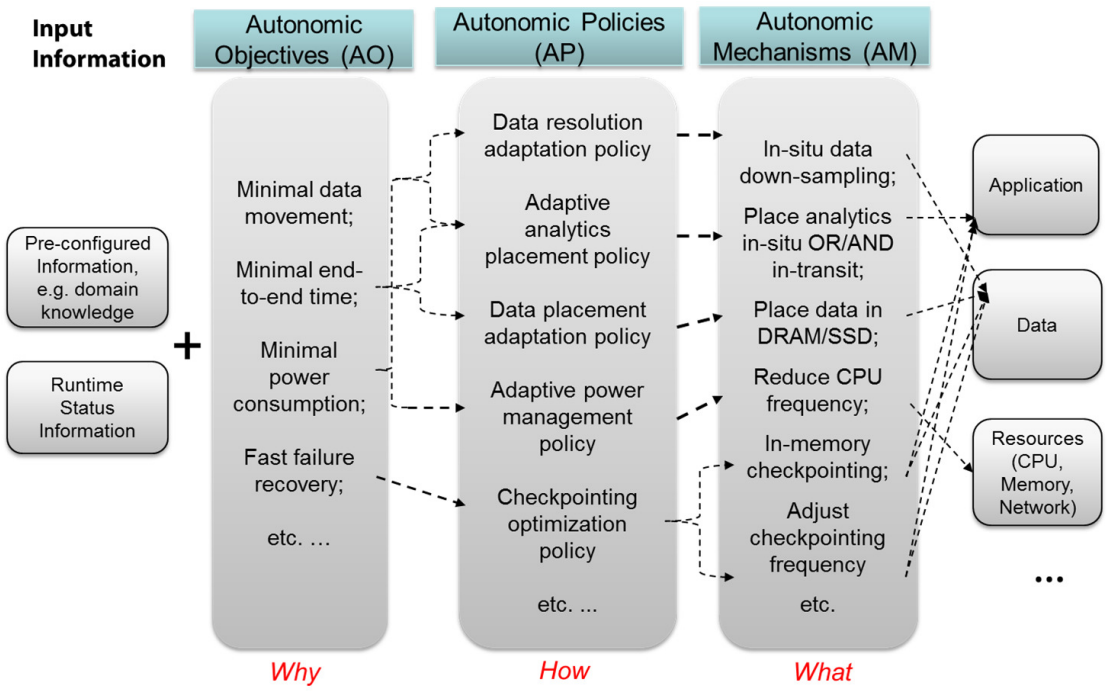



Figure 4.2: An illustration of the relationship amongst autonomic objectives, autonomic policies and autonomic mechanisms; as well as their interactions with external environment through necessary input/output.

system must be able to monitor the application behaviors and the statuses of operating environment, e.g., the runtime use of CPU, DRAM, network bandwidth, and other resources. Furthermore, the attribute of *adaptive* states that the autonomic data management runtime should be able to change the behaviors of applications and configuration of system at runtime by following specific policies. The last property is *autonomic*, which means that the triggering of adaptations should be self-managed at runtime without any other manual or external help.

This section describes the basic autonomic element used to realize such an autonomic approach, as well as how self-adaption and self-management are realized to meet the above three-property requirement.

### 4.3.1 Defining Autonomic Manager and Managed Elements

An autonomic element contains two basic components: *autonomic managers* and their *managed elements*.

**Managed Elements**: These are the smallest functional units of the application and the operating environment. The application behaviors and environment configuration can be adapted and tuned by managing these elements and controlling/adjusting their attributes. Examples of element include data objects, resource (CPU, memory, network bandwidth, etc.), application algorithms, etc. In coupled simulation workflows, the attributes of these elements, such as data locality, data quality, algorithm scalability, CPU frequency and memory allocation, impact the runtime data managements.

**Autonomic Manager**: This component provides the capability of sensing and collecting the runtime status information, and then performing the autonomic adaptations by adapting the managed elements defined above. To achieve these functionalities, the following two components are required. *Monitor* captures runtime status information from both applications and operating environment on High End Computing systems. Status information may include resource utilization and resource availability (memory, bandwidth, CPU cores) as well as application execution time, analysis time and the size of the generated data. *Autonomic Execution Engine* manages and adapts the execution of the targeted workflows to address the data management dynamism. Specifically, it has three responsibilities: (1) tuning the application and workflow behaviors, such as the placement of analytics, code coupling, etc., (2) re-configuring resources as required, and (3) managing data objects. It selects appropriate autonomic polices and then executes autonomic mechanisms based on defined autonomic objectives and status information provided by monitor.

### 4.3.2 Execution of Autonomic Adaptations

As illustrated in Figure 5.3, there are three most important stages while executing the adaptations for a certain autonomic objective. Firstly, autonomic manager detects and collects runtime status information through the monitor. After that, it selects the appropriate autonomic policies based on the autonomic objective. Based on the analysis result of the runtime information, the selected policies specify the appropriate autonomic mechanisms. Finally, autonomic manager executes these mechanisms and controls particular managed elements to finish the adaptations. These three stages compose the basic execution/control loop of

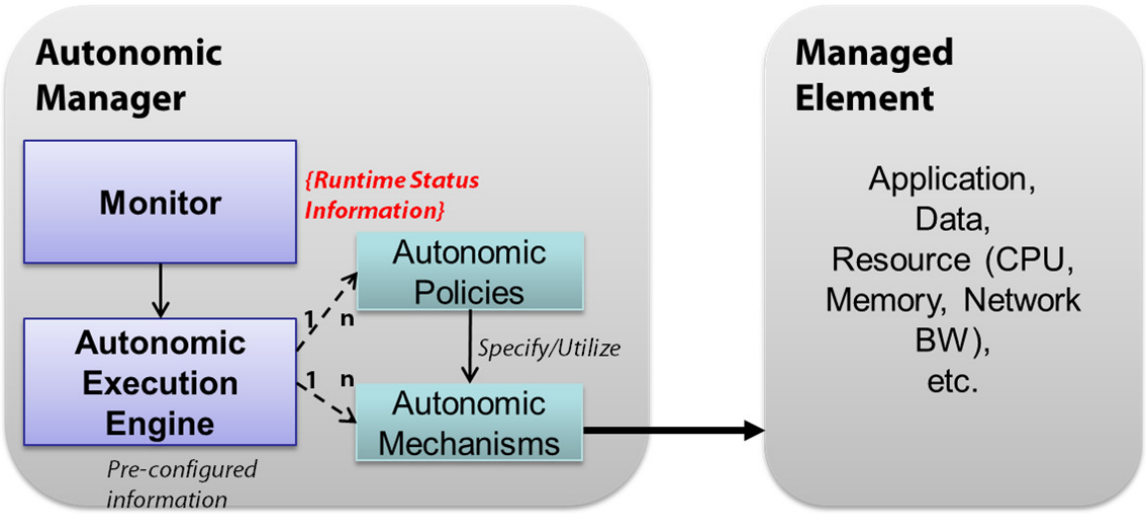



Figure 4.3: Autonomic manager and managed elements in the context of coupled scientific workflows.

an autonomic adaptation. In many coupled scientific workflows, since simulations and coupled applications are both performed iteratively, it is typically appropriate to execute the autonomic adaptation loop in each iteration.

## 4.4 Architecture of Autonomic Data Management Framework

The autonomic managements of coupled simulation workflows running on High End Computing systems require coordination among the different runtime modules of the autonomic framework. It also builds on underlying support of basic workflow and data operations, such as workflow composition, data placement (e.g., placing applications in-situ or in-transit), data transfer and message passing, etc. This section presents the architecture for the autonomic data management approach developed in this thesis. Specifically, it includes an autonomic data management module for adaptive runtime data managements, and builds on the DataSpaces framework [27] for enabling basic data transfer, storage, index and query, and coordination for coupled simulation workflows.

### 4.4.1 Autonomic Data Management Module

*Autonomic data management module* is designed and developed to support the overall execution of autonomic runtime adaptations. Figure 4.5 illustrates the layered structure of the

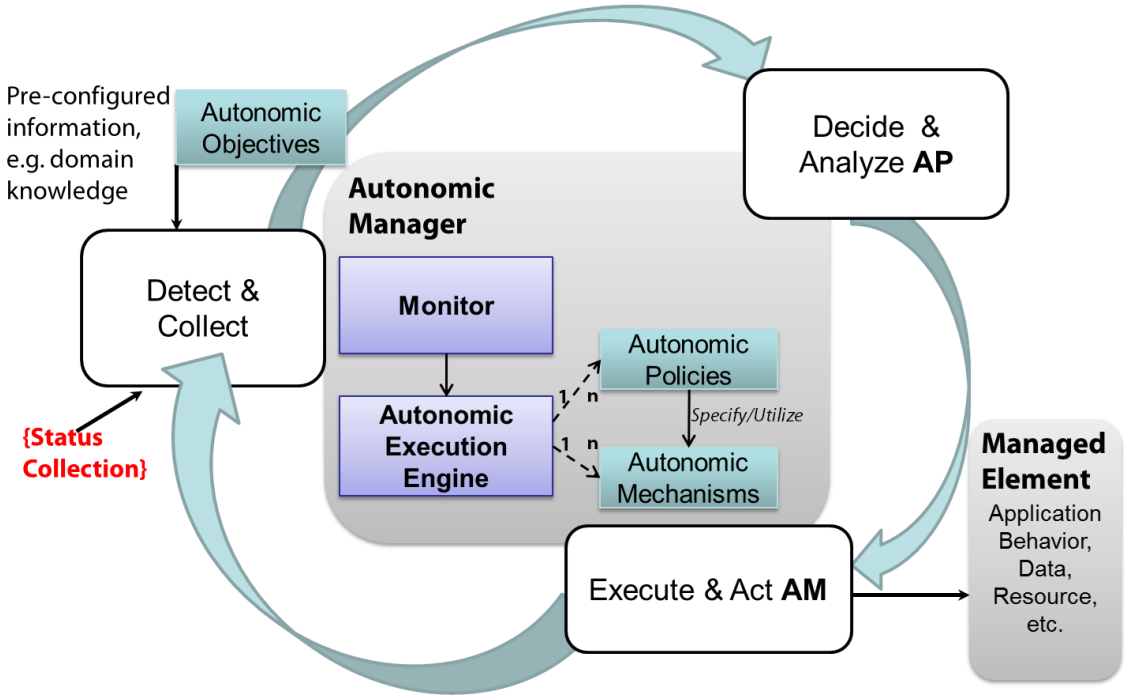



Figure 4.4: An illustration of the three-stage execution loop of autonomic adaptation.

autonomic data management module. As shown in the figure, the overall structure can be divided into three layers in vertical: application layer, middleware layer, and resource layer. The runtime status information collected in the system comes from those layers and impacts the decisions of the autonomic execution engine in terms of selecting appropriate autonomic policies and triggering corresponding autonomic mechanisms on managed elements at different layers. Moreover, autonomic objectives and pre-configured information (e.g., domain information) can be configured through specific user interfaces or configuration files.

### 4.4.2 Workflow Coordination and Data Communication Modules

DataSpaecs is a scalable data storage and sharing framework that is designed to enable dynamic code couping and interaction and coordination patterns between coupled scientific applications. Specifically, it provides a semantically tuple-based shared-space abstraction using DRAM resources of a group of staging nodes, which can be associatively accessed by the interacting applications of a simulation workflow. DataSpaces can support both in-situ data processing and in-transit data processing through static configurations.

This thesis re-uses the workflow coordination and data movement modules from the

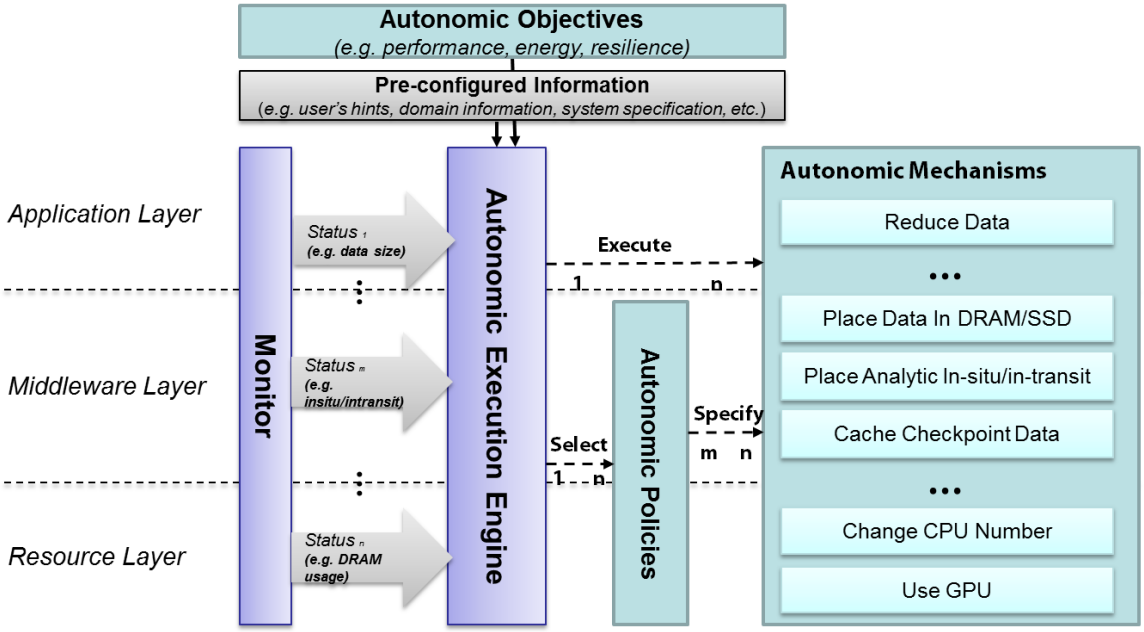



Figure 4.5: A layered architectural view of the autonomic data management module. Three layers and autonomic components are defined. Pre-configured information is user-provided information that is defined at the beginning of the job and is provided by using application interfaces or configuration files. Status information from each layer is collected as inputs into the autonomic system, to help autonomic execution engine to make runtime adaptation decisions.

DataSpaces framework [27] to realize of the autonomic data management approach. Figure 4.6 illustrates the full stack of the autonomic data management runtime architecture and how it builds on the DataSpaces framework. As it shows, the runtime realization utilizes the coordination layer of DataSpaces that provide services for asynchronous data indexing and querying, as well as its data storage layer for in-memory data caching and storage. In addition, it also leverages the data communication layer, DART [26] – an asynchronous communication and data transport substrate based on RDMA one-sided communication – to enable efficient data movement and message passing across network with low overhead.

## 4.5 Conclusion

This chapter presents an abstraction of the autonomic data management approach that can fundamentally address the runtime data management requirements in coupled simulation workflows at extreme scales. The autonomic data management approach can adaptively respond to the dynamic data management requirements and resources constraints in coupled

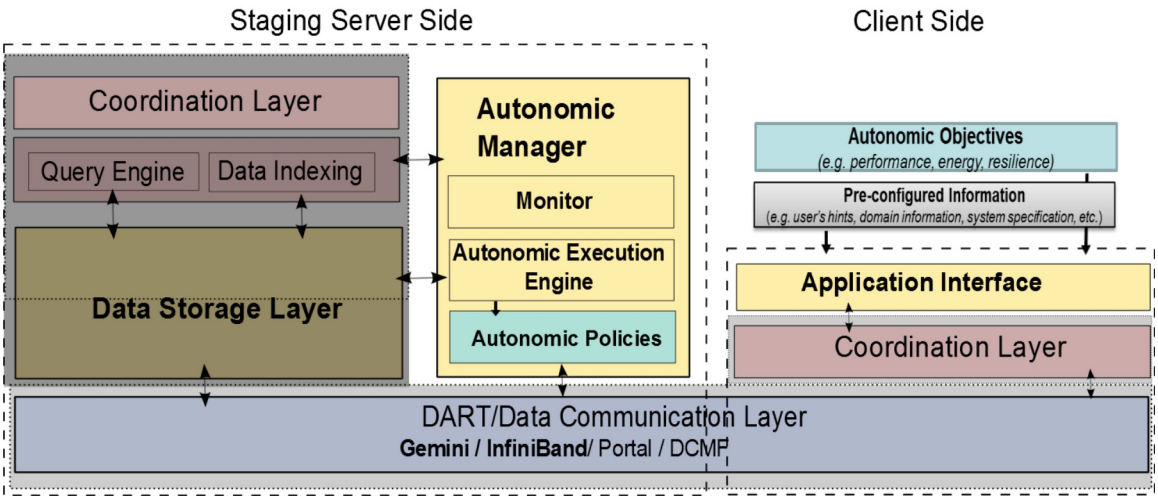



Figure 4.6: The full stack of autonomic data management framework. The shadowed area are reused part from DataSpaces.

scientific workflows.

Specifically, this chapter first presents a conceptual model that includes *Autonomic Objective*, *Autonomic Policy* and *Autonomic Mechanism* to enable autonomic data management at runtime. And then, it systematically investigated and discussed how this autonomic data management approach can be realized for coupled scientific workflows. In the realization, *Autonomic Manager* is responsible for the execution of autonomic adaptations and the controls of *Managed Elements* across different layers to achieve the adaptation objectives. Specifically, after defining a specific *Autonomic Objective*, an *Autonomic Manager* is responsible for monitoring runtime application behaviors and the status of the operating environment using *Monitor*, and managing and adapting the execution and resources of targeted workflows in a coordinated manner using the *Autonomic Execution Engine*. Finally, it presents the architecture for the autonomic data management approach that is developed in this thesis. This architecture includes an autonomic data management module for adaptive runtime data managements, and builds on the DataSpaces framework for enabling basic workflow coordination and data communication.

The next chapters will present two separate realizations of such an autonomic data management approach for different types of coupled simulation workflows.

# Chapter 5

# Autonomic In-Memory Data Management using Cross-Layer Adaptations

## 5.1 Overview

In tightly coulped scientific workflows, simulation data is quickly shared and exchanged amongst different coupled applications for accelerating the overall scientific discovery. These simulation workflows running at extreme scales are providing new capabilities and opportunities in a wide range of application areas. However, due to the scales, coupling and coordination behaviors and overall data management complexities, they are also presenting new challenges that must be addressed before their potential can be fully realized. As demonstrated in Chapter 2, many of such tightly coupled workflows contain simulations based on dynamically adaptive formulations such as Adaptive Mesh Refinement (AMR), which exhibit dynamic runtime behaviors and result in largely changed data volumes and dynamic data distributions. Efficiently managing, transporting and analyzing the simulation data in those workflows have become significant and immediate challenges.

This chapter presents an autonomic cross-layer adaption approach to address these challenges in tightly coupled workflows. It designs and pre-defines some effective adaptation policies and related mechanisms within the autonomic data management runtime system. Specifically, this approach explores the runtime adaptations at three different layers, viz., application layer, middleware layer, and resource layer. At the application layer, it dynamically performs the adaptation of spatial and temporal resolution of data being written and processed; at the middleware layer, it adapts the in-situ and/or in-transit placement and scheduling of data processing operations; and at the resources layer it adapts the allocation of in-transit resources. It also explores 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. These adaptation policies and mechanims have been implemented and integrated with an autonomic runtime and have been evaluated in terms of their effectiveness of responding to dynamic data management requirements under application objectives and resource constraints.

The autonomic cross-layer management runtime has been deployed on the Intrepid IBM BlueGene/P system at Argonne National Laboratory and the Titan Cray-XK7 system at Oak Ridge National Laboratory. With these deployments, a tightly coupled workflow that is composed of Chombo [1]-based AMR simulation and a visualization service, has been used to experimentally evaluated the behavior of the individual adaptations at each layer, as well as its effectiveness in improving overall time-to-solution, increasing resource efficiency, and mitigating I/O costs.

## 5.2 Background

As noted in Chapter 2, simulations containing dynamically adaptive formulations, e.g., AMR-based simulations, exhibit dynamic runtime behaviors and result in dynamically changing data volumes, imbalanced data distributions and heterogeneous resource (memory, CPU, network bandwidth) requirements. To illustrate the dynamic data management and processing requirements in such workflows, this section considers the *3-D AMR Polytropic Gas* application as an example, which 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 5.1 plots parts of a profile of the distribution of the application's peak memory usage on 4000 CPU cores over 50 time steps. As it shows, memory consumption varies significantly both across cores and over time. Specifically, besides memory consumption increases for each time step, the pace in which the memory consumption increases is erratic. Moreover, the memory usage is not distributed evenly among these processes. These characteristics of dynamic runtime behaviors 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

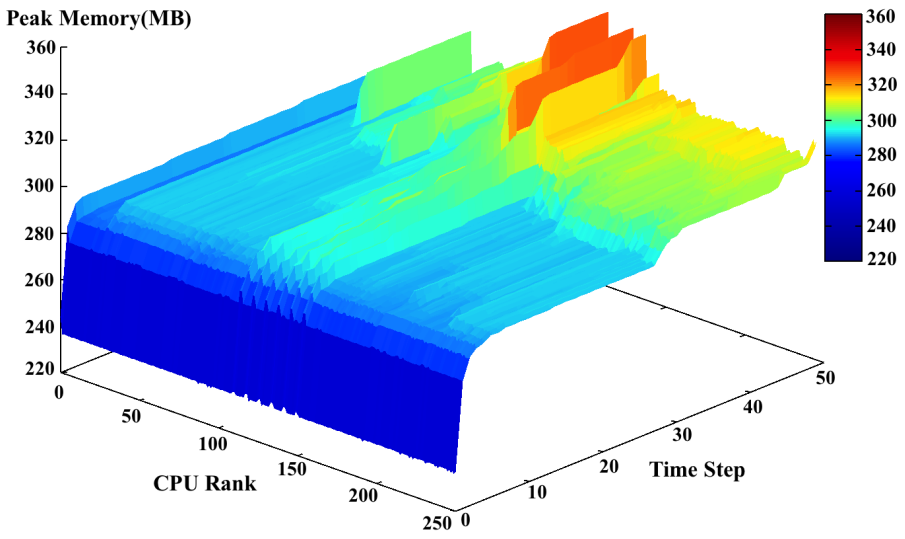



Figure 5.1: Distribution of the peak memory consumption for an AMR-based Polytropic Gas simulation using the Chombo library.

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. This chapter explores how these dynamic adaptations can be realized at runtime for AMR-based simulation workflows on large-scale systems. It also presents the exploration of policies and mechanisms for combining these adaptations in a coordinated and cross-layer manner to better address application requirements and constraints.

## 5.3 Realizing Cross-Layer Adaptations

This section describes an 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 autonomic execution engine, and autonomic policies, as illustrated in Figure 5.2 and described below. In this approach, users can provide two types of inputs. *Autonomic objectives* define the objectives that users expect to achieve, such as minimizing time-to-solution, minimizing data movement, using highest available data resolution, etc. *User's 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 objectives and hints, the operational state provided by the monitor, and adaptation policies.

Three adaptation mechanisms are explored in this chapter. 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

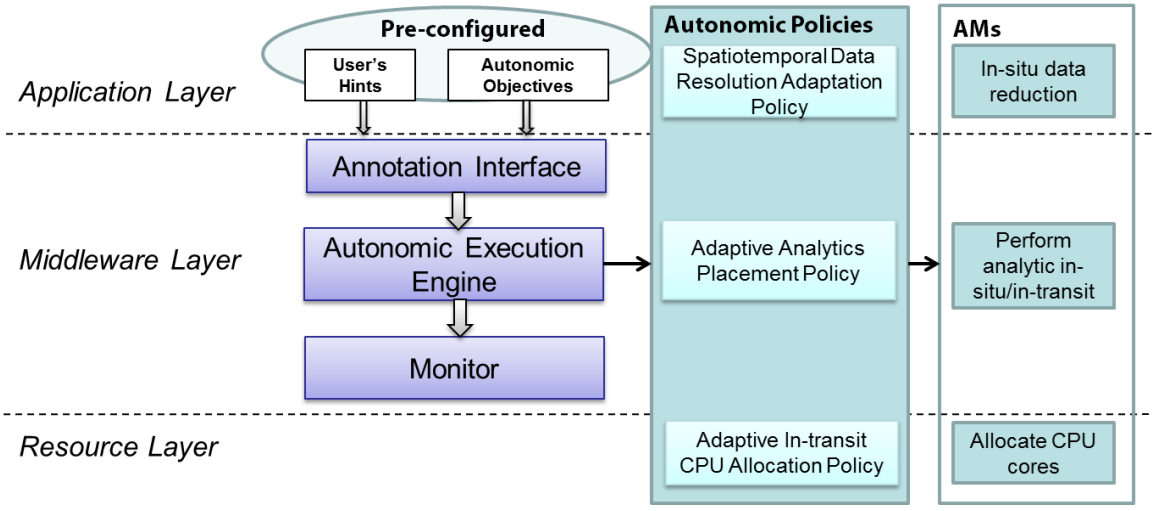



Figure 5.2: A conceptual architecture of an autonomic approach for realizing runtime adaptations for in-situ/in-transit implementations of coupled simulation workflows.

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 necessary.

The *Adaptation Policies* specify which adaptation mechanism(s) should be executed based on user inputs and the operational state. The following Section presents the development of 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 5.3. 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).

## 5.4 Defining Adaptation Policies

This section presents a development of adaptation policies for an AMR-based simulation workflow. Note that rather than finding optimal adaptations, the overall goal is to develop

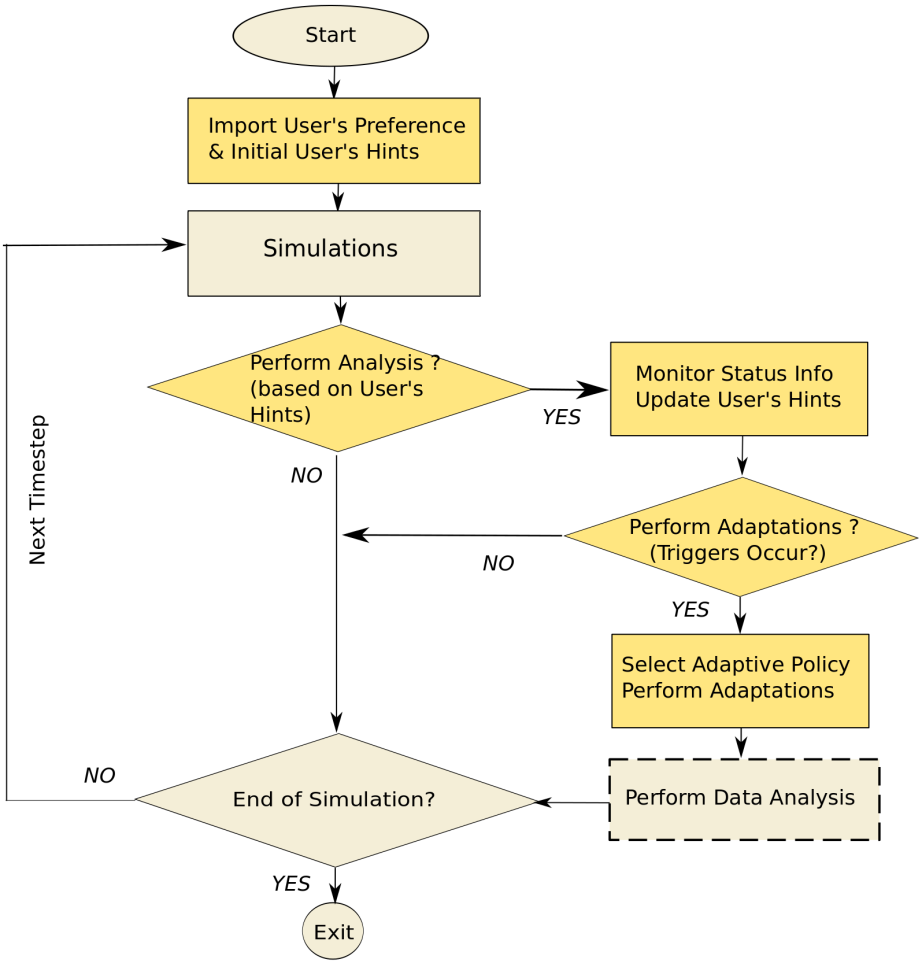



Figure 5.3: An overview of the autonomic adaptation process.

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

Table 5.1: Notation used in formulating adaptation policies.

policies that can be efficiently and scalably implemented at runtime on very large scale system while satisfactorily addressing application requirements/constraints. Specifically, it develops 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 5.1 summarizes the notation used in this discussion.

### 5.4.1 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 resolution that can be effectively processed in-situ or transferred to in-transit resources given autonomic objectives 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 $i^{\text{th}}$ simulation iteration is determined by the following policy:

Maximize

$$S_{data} - f_{data\_reduce}(S_{data}, X) \tag{5.1}$$

Subject to

$$Mem_{data\_reduce}(S_{data}, X) \leqslant Mem_{available} \tag{5.2}$$

(memory requirement)

$$when X \epsilon \{X_1, X_2, \cdots, X_n\} \tag{5.3}$$

(set of acceptable down-sample factors)

### 5.4.2 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 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 5.4 and are expressed in the formulations below:

Since

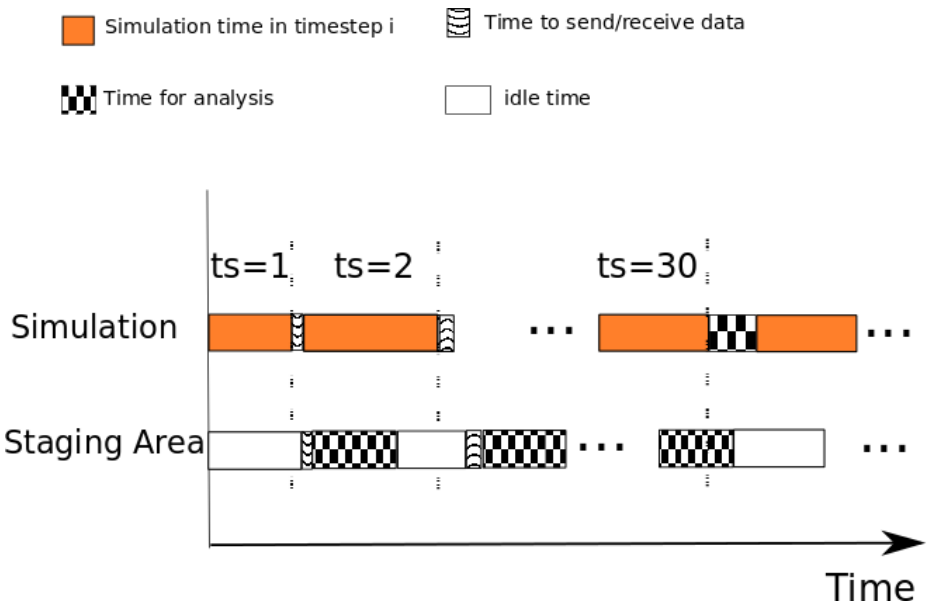



Figure 5.4: Illustration of the analysis placement adaptation policy. For adaptation at ts=1 and 2, in-transit resources are idle, and as a result analysis is placed in-transit. For adaptations at ts=30, since in-transit resources are busy, the analysis time for in-situ and in-transit processing are 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.

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

$$+\bar{D_i} \cdot (T_{i\_insitu\_wait})\} \tag{5.4}$$

$$T_{sum\_intransit} \simeq \sum_{i=1}^{ITER}\{\bar{D_i} \cdot T_{i\_intransit}(M, S_{i\_data})$$

$$+T_{i\_intransit\_wait}\} \tag{5.5}$$

Minimize

$$max\{T_{sum\_insitu}, T_{sum\_intransit}\} \tag{5.6}$$

(minimized time-to-solution)

Subject to

$$\bar{D_i} \cdot (T_{j\_intransit\_remaining}(M, S_{j\_data}) < T_{i\_insitu}(N, S_{i\_data}))$$

$$= 1, j < i; \tag{5.7}$$

(execution time estimation)

$$D_i \cdot (Mem_{intransit}(S_{i\_data}, M) < S_{data}) + \bar{D_i} \cdot (Mem_{available}$$

$$\leqslant Mem_{insitu}(S_{i\_data}, N)) = 1 \tag{5.8}$$

(resource constraints).

### 5.4.3 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 $i$th 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

$$T_{i+1\_sim}(N) + T_{i+1\_sd}(S_{i+1\_data}) = T_{i\_intransit}(M, S_{data}) + T_{i\_recv}(S_{i\_data}) \tag{5.9}$$

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

and

$$Mem_{intransit} > S_{data} \tag{5.10}$$

(in-transit memory constraint)

### 5.4.4 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, this section presents 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 the 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 this 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 this 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 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*.

## 5.5 Experiment Evaluation

This section presents an experimental evaluation of the autonomic data management framework presented in this chapter. It first evaluates adaptations at each of the three layers individually, and then evaluates combined cross-layer adaptations.

### 5.5.1 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:** These experiments 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 [55, 84], 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. This algorithm 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 prototype of Autonomic Runtime

The autonomic runtime is implemented on the top of DataSpaces data-management substrate [8, 28, 27]. 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*.

The autonomic approach has been implemented and integrated with Chombo. In particular, only minimal modification is needed for the simulation side. In the `AMR` class of Chombo, a virtual function named `AMR::insituProcess` is added and called periodically

during `AMR::run`. The `AMR::insituProcess` function can be implemented in `AMR`'s derived classes for each particular case. Through a derived class, it can customize the in-situ processing tasks, access simulation data, and minimize the interference to Chombo. Moreover, the Chombo's MPI communicator can be accessed simply through the global variable `Chombo_MPI::comm`. During the MPI initialization stage, the processors are partitioned into three groups for simulation and in-situ processing, data transferring, and in-transit processing, respectively. This way can further minimize the possible interference between the simulation and the other processing. This approach is easy to understand and simple to integrate with the exiting real-world large simulations.

#### Systems

The 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 they are not used in experiments.

### 5.5.2 Evaluation and Discussion

#### Evaluation of Adaptations at the Application Layer

These experiments used a 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, the experiments were performed 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

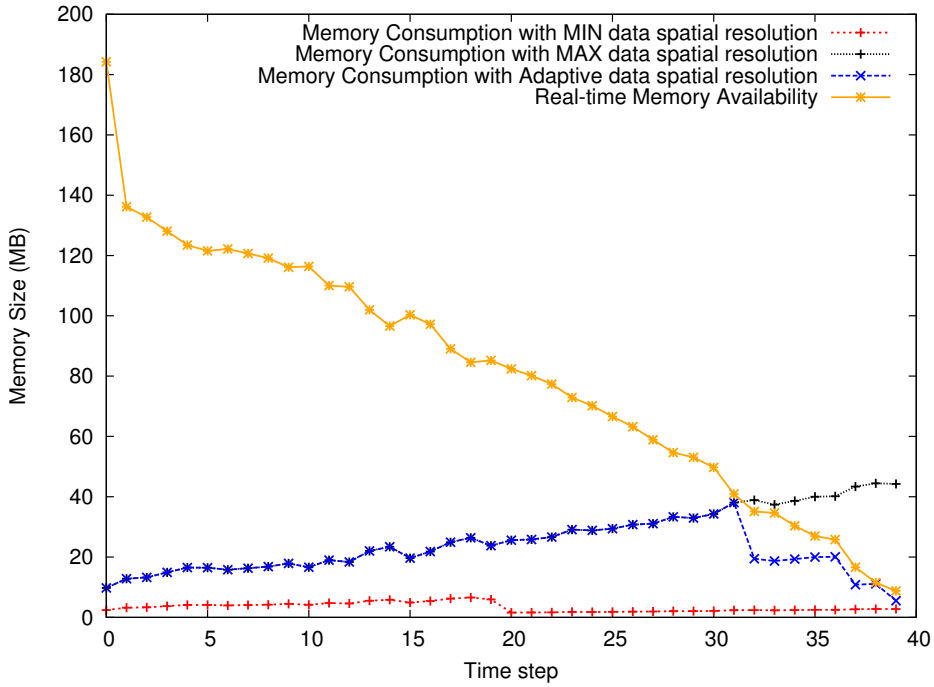



Figure 5.5: Evaluation of application layer adaptation of the spatial resolution of the data using user-defined downsampling factors, and based on runtime memory availability. Note that at the 31st time step, the spatial data resolution is reduced due to limited availability of memory resources, and at the 40th time step, the data resolution reaches the minimal value.

of down-sampling factors. These ranges of acceptable down-sampling 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.

In this experiment, the peak memory used on a processor varied from $20MB$ to $>$ 300MB. Figure 5.5 plots the runtime 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 users with a theoretical framework to measure the information content of a variable [82]. For each data block within the AMR dataset, it computes the *entropy* value to quantify the distribution of its variables. For a discrete random variable $\chi$ and probability mass function $p(x), x \in \chi$, the entropy of $X$ can be defined as

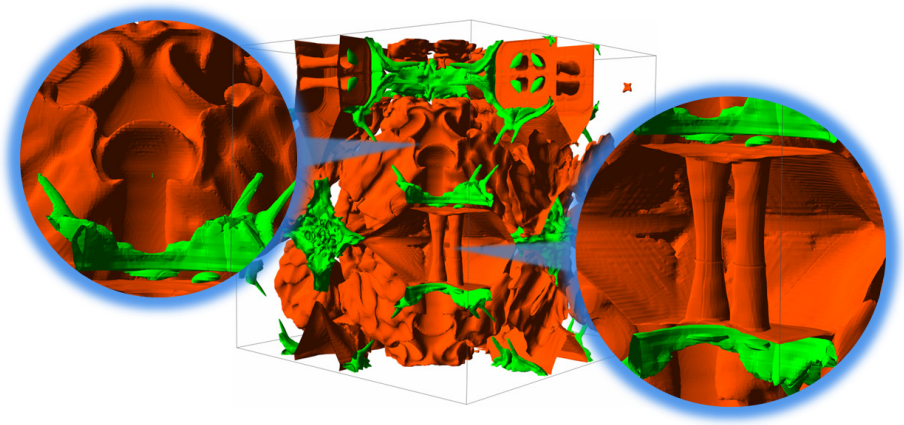
$$H(X) = -\sum_{x \in \chi} p(x) log p(x) \tag{5.11}$$

where $p(x) \in [0, 1]$, $\sum_{x \in \chi} p(x) = 1.0$, and $-log p(x)$ represents the information associated with a single occurrence 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. Therefore, the data blocks can be adaptively downsampled based on their entropy values by specifying a set of thresholds. Figure 5.6 compares visualizations using the full-resolution data and the adaptively down-sampled data. It can be seen 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 the understanding of the data.

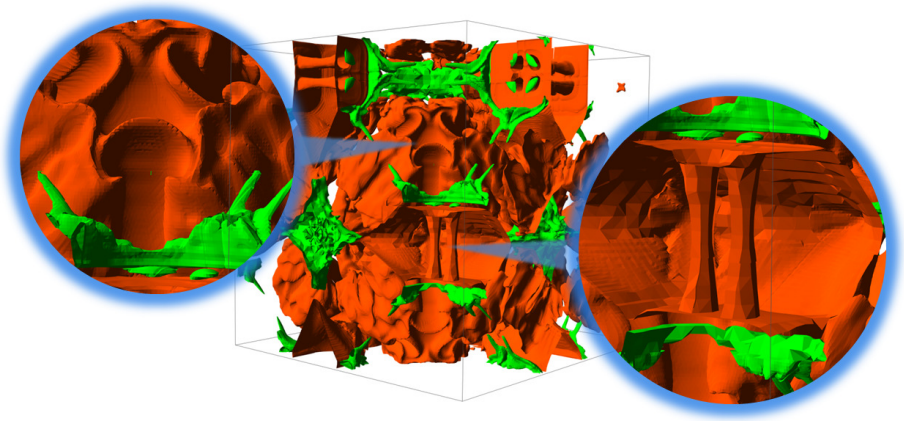
These results clearly show that the proposed approach successfully adapts the down-sampling factor at runtime to meet the constraints on acceptable data resolution at the application layer as well as constraints due to the limited size of available memory at the resource layer. The results also show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

### Evaluation of Adaptations at the Middleware Layer

These experiments 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. The simulation were performed on 2K, 4K, 8K and 16K cores, with a 16:1 ratio of the number of the simulation



(a)



(b)

Figure 5.6: 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.

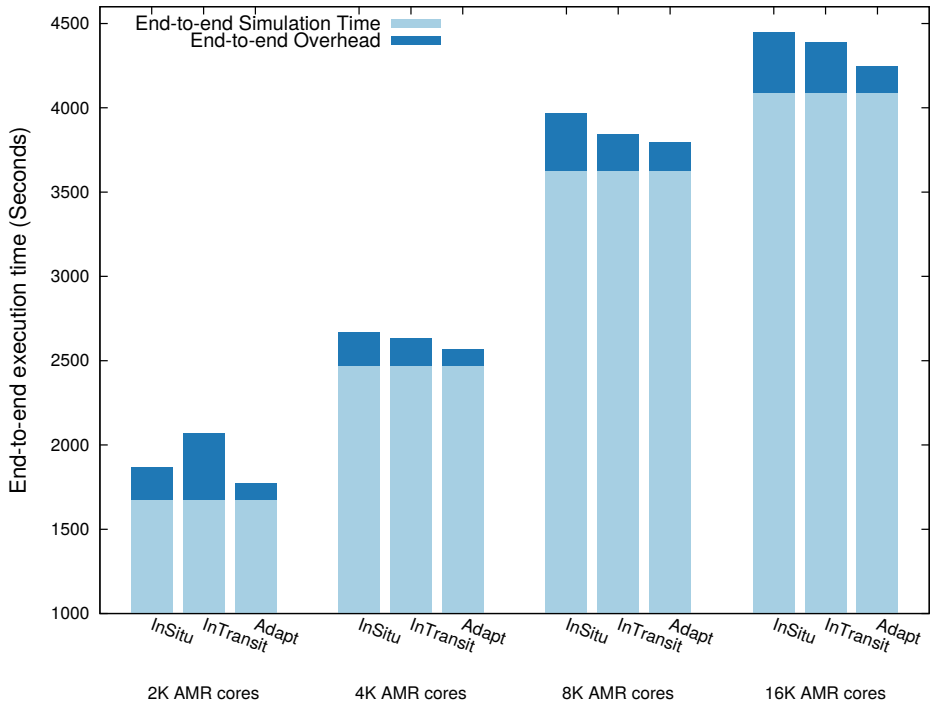



Figure 5.7: Comparison of cumulative end-to-end execution time between static and adaptive in-situ/in-transit placement of the visualization service. The end-to-end overhead plotted is the overhead on the overall time-to-solution and includes data processing time, data transfer time, and other system overheads.

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 performance evaluation and is composed of two components as seen in Figure 5.7: *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 this policy achieves its 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

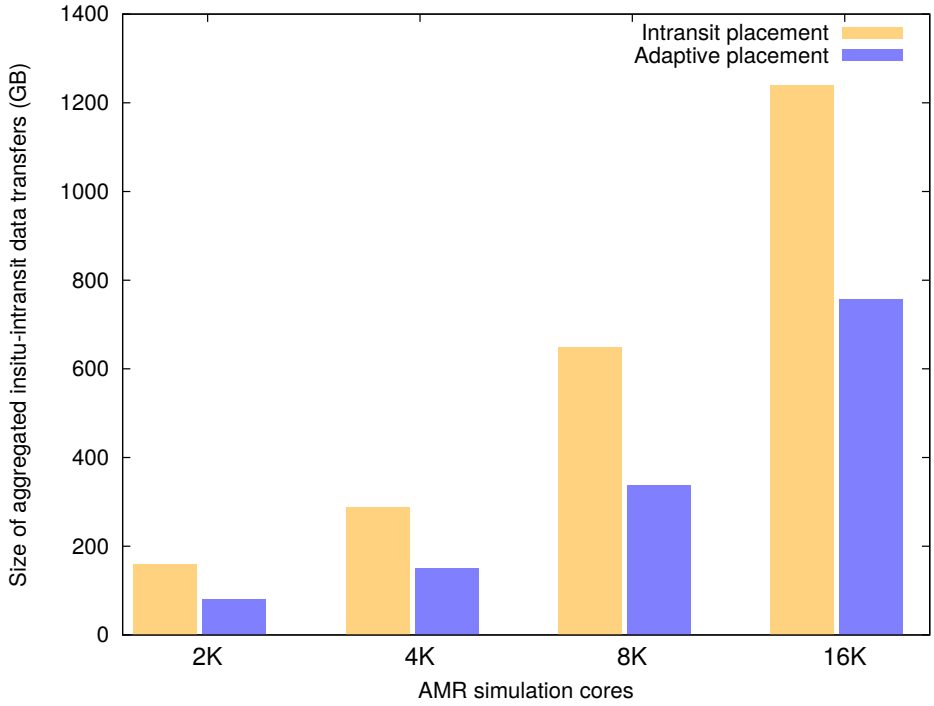



Figure 5.8: Comparison of total data movement between static and adaptive in-situ/in-transit placement of the visualization service.

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%, 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 5.8.

### Evaluation of Adaptations at the Resource Layer

This experiment 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 5.5.2. 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 5.9 plots the number of in-transit cores used at each time step. At the beginning

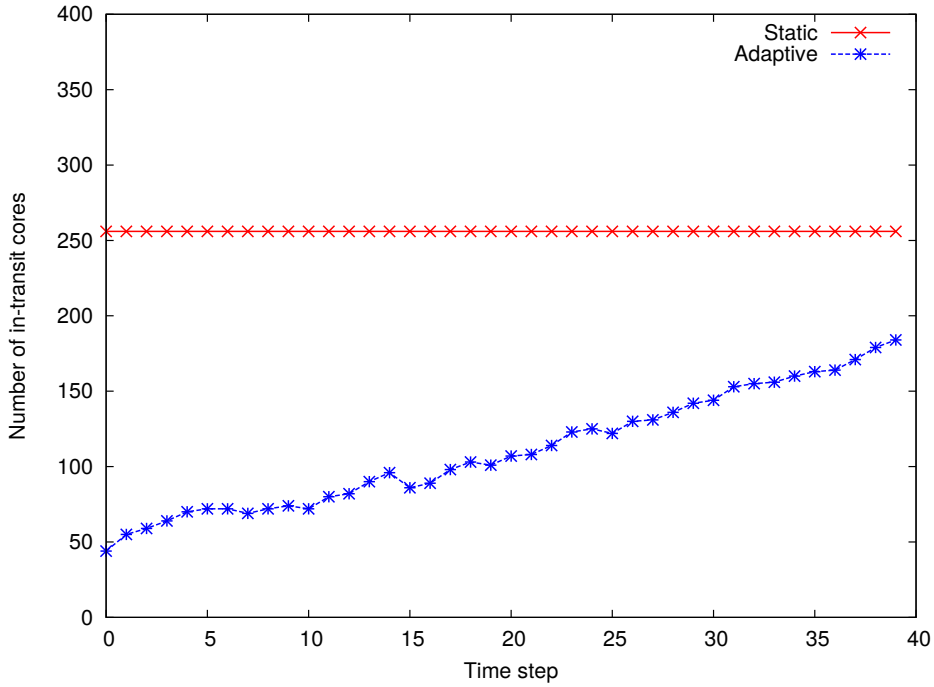



Figure 5.9: Number of in-transit processor cores performing analysis when using resource layer adaptations.

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, the cpu utilization efficiency can be defined as follows:

$$\frac{\sum_{j=1}^{TS}\sum_{i=1}^{M_j}\{T_{intransit\_analysis\_i\_j}\}}{\sum_{j=1}^{TS}\sum_{i=1}^{M_j}\{T_{intransit\_total\_i\_j}\}} \tag{5.12}$$

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

Using this definition of utilization efficiency, it can be found that the utilization efficiency

| Cases | Total Time Steps | No. of Time Steps for Different Utilizations | | | |
|---|---|---|---|---|---|
| No. of Sim Cores : No. of Staging Cores | | 100% Cores | 75% Cores | 50% Cores | < 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 5.2: Actual utilization of in-transit cores while performing in-transit analysis.

when using resource layer adaptations is 87.11% as compared to only 54.57% in the static allocation case.

### 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 5.5.2. The experiment also used the same acceptable user-defined data sampling rates that were used in the experiments described in Section 5.5.2, 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 5.10 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 5.5.2. Since the data is reduced in-situ using downsampling, the time required for in-situ analysis and in-transit

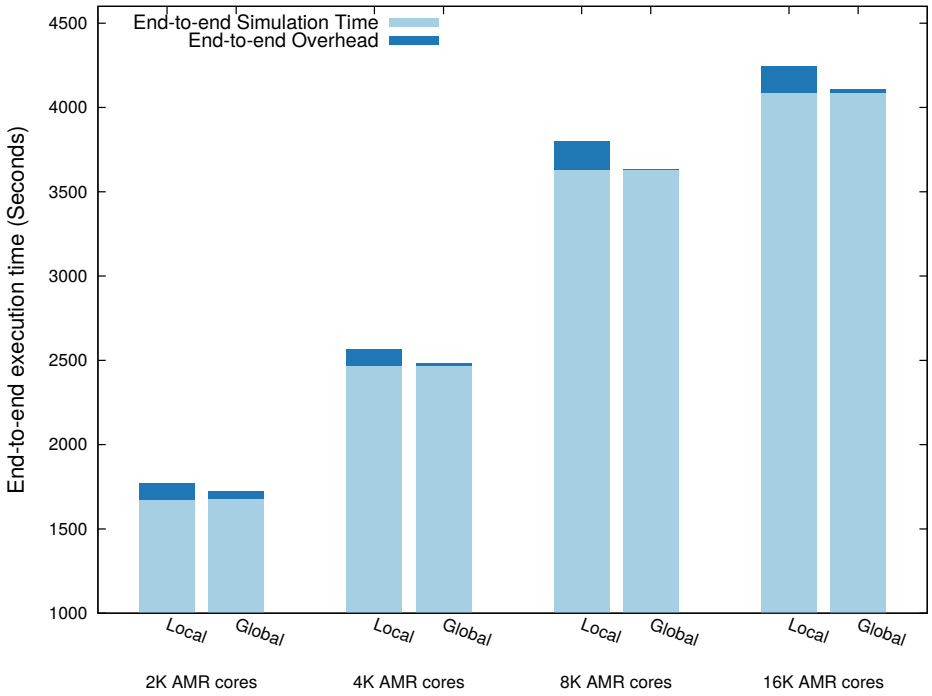



Figure 5.10: Comparison of cumulative end-to-end execution time between for combined cross-layer adaptations (i.e., global adaptations) and middleware layer only adaptations (i.e., local adaptations).

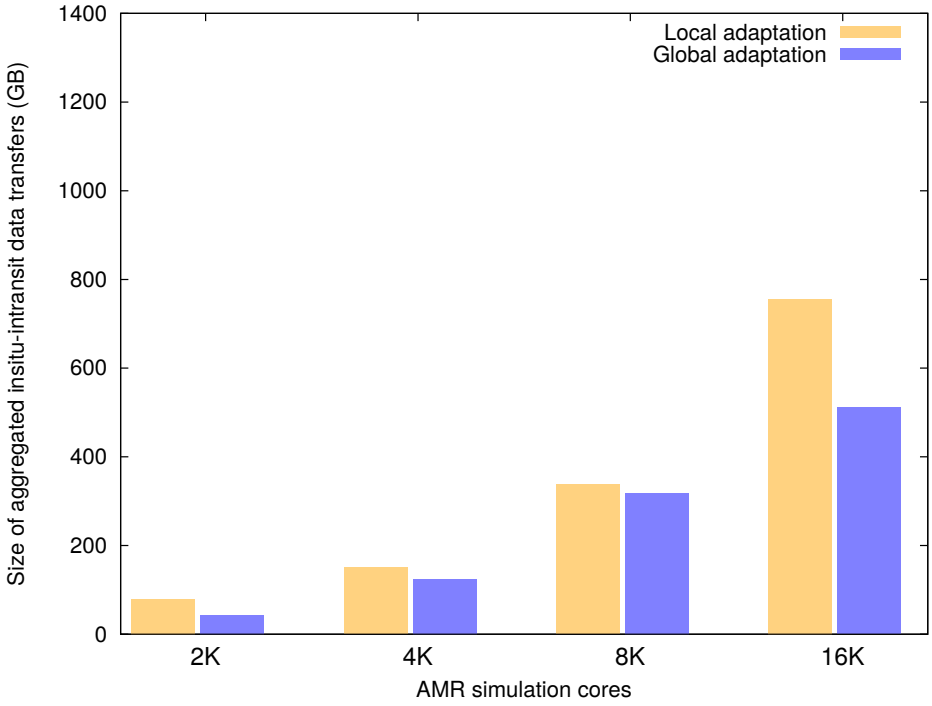



Figure 5.11: Comparison of the total data movement for combined cross-layer adaptations (i.e., global adaptations) and middleware layer only adaptations (i.e., local adaptations).

analysis decreases due to the decreasing data volume. On the other hand, faster in-transit 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 5.2.

While performing more of the analysis in-transit implies a larger amount of data transfer, Figure 5.11 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 5.5.2. Meanwhile, Table 5.2 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, the 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 the experiments use Chombo-based applications and a third-party adjustable visualization code, the cross-layer autonomic 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.

## 5.6 Conclusion

This chapter explored autonomic 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, it 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. 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 in terms of reducing network data movement, improving resource utilization and minimizing time-to-solution.

# Chapter 6

# Autonomic Data Placement in Multi-Tiered Data Staging

## 6.1 Overview

Architectural trends indicate that emerging systems will have increasing numbers of cores per node and correspondingly decreasing amounts of DRAM memory per core as well as decreasing memory bandwidth. These trends can significantly impact the effectiveness of in-memory staging solutions and their ability to support data-intensive simulation workflows. For example, in a loosely coupled data-intensive simulation workflow, data generated by one simulation may have to be consumed by another simulation or by an analysis service before it can be overwritten. As data volumes increase and memory capacity and bandwidth decrease, memory constraints can quickly become a bottleneck of in-situ/in-transit processing. Furthermore, the coupled simulation, the analytics, and/or the visualization components may require the accumulation of data over multiple time steps, which can further accentuate this problem in such a type of coupled simulation workflows.

Fortunately, non-volatile memory devices, such as solid state disks (SSDs), are becoming more pervasive and have been deployed on a number of systems, such as Gordon at San Diego Supercomputing Center (SDSC), Tsubame2 at Tokyo Institute of Technology, and Sith at Oak Ridge National Laboratory. SSDs offer several benefits over traditional hard disks due to their lower data access latency, lower power consumption, and stability. Furthermore, these lower costs and larger capacities as compared to DRAM make the SSD an attractive candidate as an intermediate data storage level to address the performance and latency gaps between DRAM and magnetic disk [87]. As a result, several existing HPC research efforts explore using SSDs, for example, as storage buffers to temporally cache checkpoint data [71], or for caching data before it is stored onto disks [49].

However, additional complexities associated with managing placement of data across

multiple layers of the memory hierarchy and its access by multiple concurrently executing tasks present significant challenges. As a result, these intermediate memory levels have not been extensively used for supporting in-situ/in-transit data staging and processing for coupled simulation workflows. In fact, system support and runtime mechanisms for dynamically managing data placement across the layers of a deep memory hierarchy, coordinating data movement and data sharing between the components of a coupled simulation workflow remain research challenges.

This chapter explores how a SSD-based level of deep memory hierarchy can be used for data staging, coupled with the autonomic data management approach to address the challenges outlined above in loosely coupled workflows. Specifically, it presents a multi-tiered data staging runtime that leverages both DRAM and SSD to support dynamic data staging and runtime data management for coupled data intensive simulation workflows. The presented hybrid staging approach allows us to accommodate larger data volumes that may exceed available DRAM main memory within the staging nodes, and to support runtime data sharing, code coupling, and data management required by simulation workflows. It also presents an adaptive application-aware data placement mechanism that can dynamically manage and optimize data placement across the DRAM and SSD storage layers using data access patterns and user provided hints. More specifically, applications can specify and update spatial and temporal attributes describing their expected data access patterns as hints along with the workflow. At the same time, data access patterns are tracked at runtime and are used to anticipate future accesses. The adaptive runtime uses both the users hints and the runtime predictions to assess the access probabilities and importance of each data object, and then quantifies these data access possibilities and importance by calculating and assigning a utility value. Data objects with higher utility value will be prefetched from a lower level of the memory hierarchy (i.e., SSD) to a higher one (i.e., local DRAM), and/or stay at higher memory hierarchy level longer, which is demonstrated to reduce the concurrent data access overheads and improves the overall time to solution.

This data staging runtim has been implemented and deployed on two different systems at the Oak Ridge Leadership Computing Facility (OLCF): the Sith Infiniband system and the Titan Cray XK7 system. Since Titan is not equipped with SSDs, a simple emulator is used,

which utilizes part of the available DRAM to mimic data write/read behaviors to SSD, and models the performance of SSDs by introducing delays. These implementations are designed and used to experimentally evaluate the behavior and performance of the staging runtime using two application workflows: the S3D[13] (DNS-LES) combustion simulation workflow and the XGC1-XGCa confined high-temperature plasma fusion simulation workflow [20, 44].

## 6.2 Data Management Challenges

As scientific simulations grow in size, the efficient management of the data they generate becomes imperative in order to enable scientists to glean insights from them in a timely manner. This section describes the specific data challenges present in today's scientific workflows.

Figure 5.2 illustrates a coupled simulation workflow, wherein the primary scientific simulation is the data producer and secondary simulations, analytics, and/or visualization applications serve as data consumers coupled to the producer. These components run at different scales, produce and consume data at different rates, and progress at different speeds. They also regularly exchange data at runtime with differing exchange patterns, frequencies, and data sizes. The key challenges are then to manage the runtime placement of this data and to optimize the end-to-end performance of the workflow by orchestrating an ensemble-based execution of the various components. This approach must also account for the complexities of the data exchanges that will occur between each of the components.

In-memory data-staging approaches are addressing these challenges, i.e., previous work DataSpaces, which provides a semantically specialized shared-space abstraction and underlying runtime mechanisms to support in-situ/in-transit workflows using compute and storage resources at a set of staging nodes, as well as asynchronous data exchange via RDMA [93, 28].

Despite the advantages that in-memory staging solutions offer, increasing data volumes, as well as non-uniform input/output staging rates and varying data sizes, can still result in situations where data no longer fits within the local DRAM staging area. Furthermore, some analytic components, e.g., feature tracking[47] and trajectory tracking visualizations[86], may require data generated at multiple simulation time steps. This results in a large amount

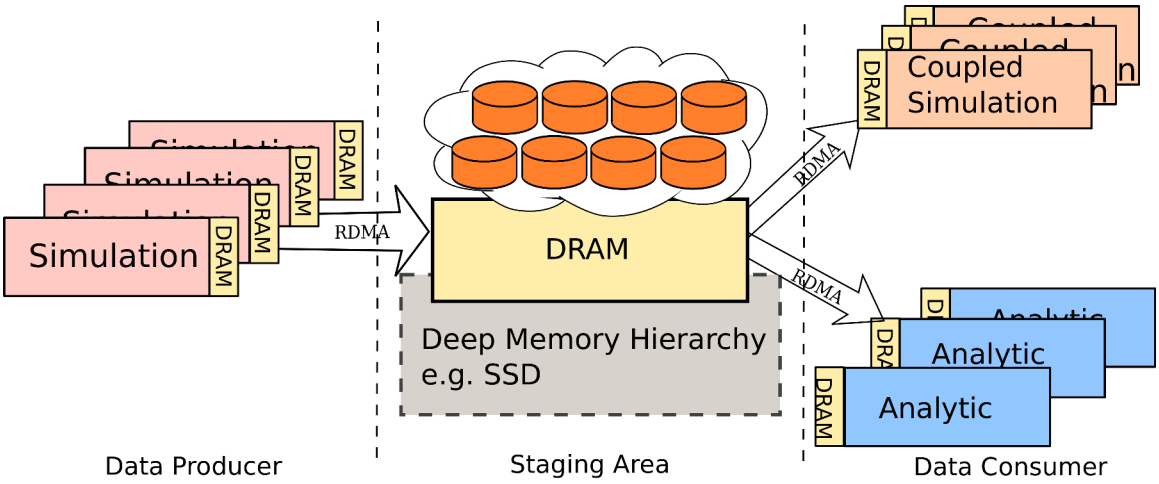



Figure 6.1: A typical data staging workflow to enable coupled simulation/analytics components.

of data that would have to be cached in the staging area, but DRAM capacity is limited on these computational resources. Recent architecture trends suggest that future high end computing resources will likely add more cores and compute power, while decreasing the amount of DRAM per core; therefore, solutions that only aim to allocate additional resources to the staging area are not feasible or desirable, and new approaches must be considered.

## 6.3 Design of the Multi-tiered Staging Framework

The overarching goal of this part of research is to develop an efficient runtime staging framework that can leverage multiple levels of the memory hierarchy to support data intensive coupled simulation workflows. Specifically, this approach extends existing staging solutions vertically by spanning two levels of local memory resources – DRAM and SSD. The resulting multi-tiered staging framework allows us to accommodate larger data volumes that may exceed available DRAM memory in the staging area, and to support runtime data sharing, code coupling, and data management required by data-intensive simulation workflows. This research also includes a development of an adaptive application-aware runtime mechanism to manage data placement across the memory levels and orchestrate data movement. More specifically, this approach uses data access patterns and user provided hints to assess the

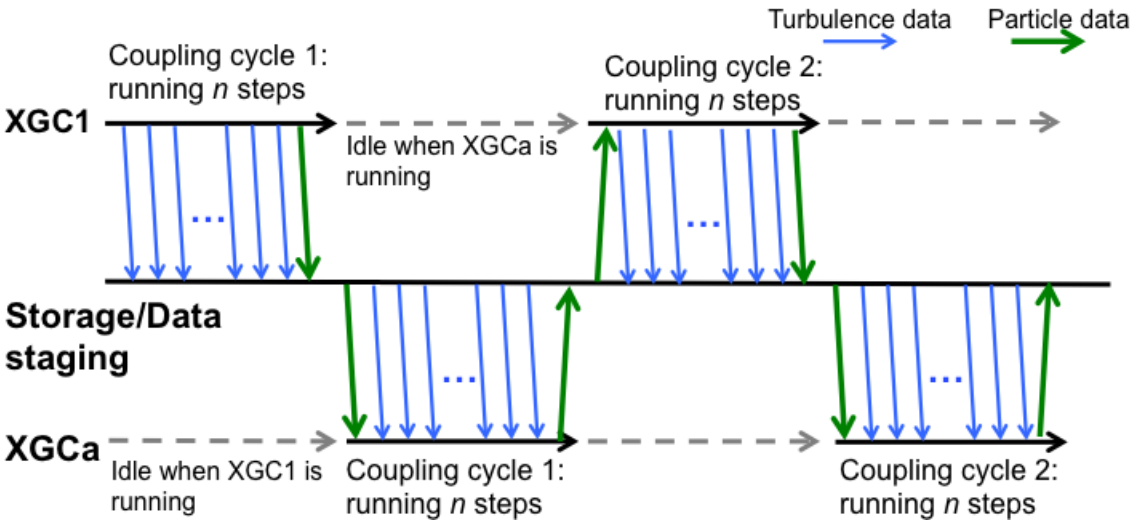



Figure 6.2: Coupling and data-exchange pattern for the XGC1-XGCa coupled simulation workflow. Two types of data is exchanged between XGC1 and XGCa in each coupling step.

utility of data objects and appropriately place the data in the multi-tiered staging area.

A schematic overview of the architecture of the multi-tiered staging framework is presented in Figure 6.3. The framework builds on DataSpaces [28], and directly leverages its *Coordination Layer* and *Data Communication Layer*, reusing its data transport, indexing and querying capabilities. Following the DataSpaces architecture, the framework consists of a client side subsystem that is co-located with the workflow applications and a server side subsystem that runs on the staging cores. The key modules of the multi-tiered staging framework, *Application Interface*, *Data Object Storage Layer* and *Data Object Management Module*, are described below.

### 6.3.1 Application Interface

The spatial and temporal data domain, which is typically based on a discretization of the physical domain, is widely used in scientific applications to specify data regions of interest as well as the frequency of data processing. This framework uses this information to assist in data placement. Specifically, a set of APIs on the client side are defined through the *Application Interface* to enable users to explicitly annotate the workflow specification with *hints* in the form of spatial and temporal attributes describing their expected data access

patterns. These *hints* are transferred to the staging server and decoded by the *Data Object Management Module* to help optimize data placement. The specific APIs are described in the next section.

### 6.3.2 Data Object Management Module

The *Data Object Management Module* implements both the multi-tiered staging mechanism at the staging servers and the data placement within the staging area. Specifically, it implements the application-aware data placement mechanism that uses hints along with history-based predictions of data access patterns to assess the probability of accessing data objects in the staging area. Once this probability is assessed, it can prefetch data objects accordingly from a lower level of the memory hierarchy, i.e., SSD, to local DRAM, thus reducing access overheads and improving the overall time to solution. It is also responsible for managing the versions of data stored in the staging area and the associated metadata, and handling garbage collection along with other utility functions.

### 6.3.3 Data Object Storage Layer

The *Data Object Storage Layer* spans the primary DRAM storage level, which has relatively limited capacity but lower latencies, and the secondary SSD storage level, which has relatively larger capacity but higher latency. Data objects are appropriately stored at different storage levels across distributed staging nodes as defined by the *Data Object Management Module*.

## 6.4 Application-aware data placement mechanism

During the execution of a simulation workflow using a staging-based approach such as DataSpaces, at the end of a time step, the simulation issues a data write request and then proceeds with its next time step. Meanwhile, data produced by the simulation is asynchronously transported to the staging nodes using RDMA. Therefore, the progress of the simulation is not impacted by whether the data is placed in DRAM or SSD at the staging nodes. However, a delay in data retrieval can directly impact the execution of a coupled simulation and/or analytics component, and may significantly increase the overall

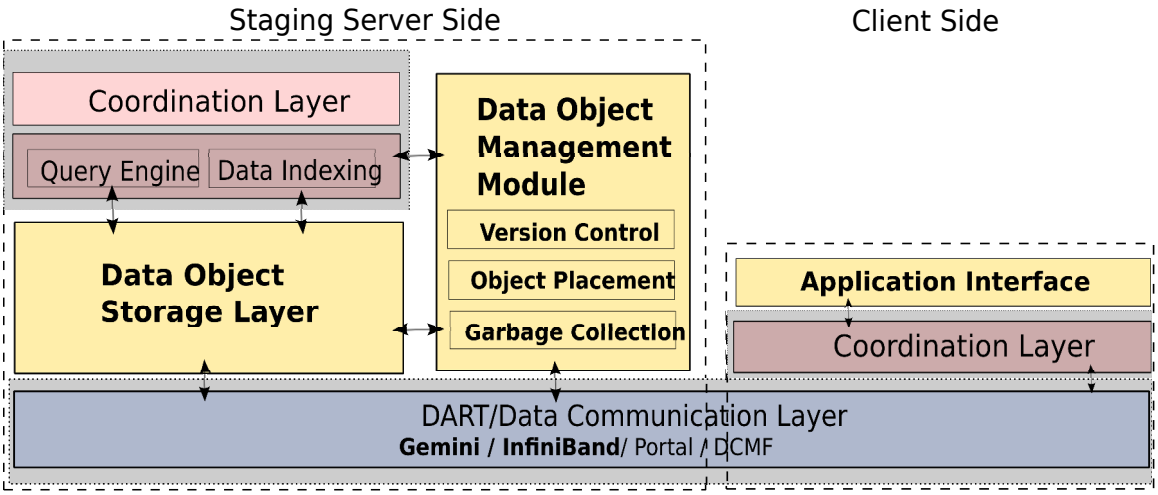



Figure 6.3: System architecture of multi-tiered staging framework. The shadowed area represents components of DataSpaces that are reused by the framework.

execution time of the workflow. As a result, an adaptive application-aware data placement mechanism is designed, which focuses on moving data across the levels of the memory hierarchy at the staging nodes in order to optimize data retrieval requests. Generally, it attempts to ensure that data is moved to DRAM prior to a read request from a coupled simulation or analytics component in order to avoid any penalty from reading data from a lower level of the memory hierarchy, i.e., the SSD level.

To achieve this goal, this research quantifies the utility (or value) of data objects in the staging area based on anticipated data read patterns. If a specific data object is part of a possible future data read patterns, its *data utility value* increases. Data objects with higher *data utility value* are kept longer in the staging area and have a greater chance of being moved to a higher level in the memory hierarchy. Data read patterns are anticipated using information from two sources: (1) domain information and hints about expected data read patterns provided by the user along with the workflow, and (2) the predictions from an adaptive data read pattern predictor that utilizes both application-level data locality and data access history. The data read pattern predictor is designed to address inaccuracies in the user-provided hints (e.g., the observed reading patterns conflict with the provided hints) and to cases where there is no information about reading patterns provided by the user (e.g., in case irregular data read patterns). Details of the adaptive application-aware

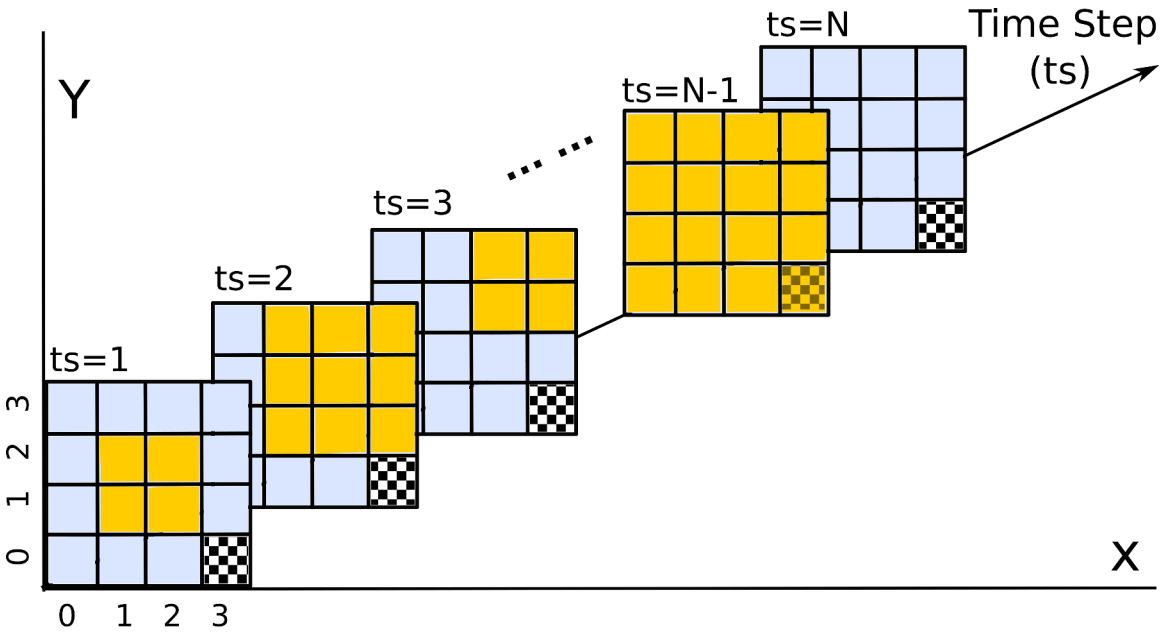



Figure 6.4: An illustration of spatial and temporal data read patterns for a 2D data domain with N time steps. The gray regions indicate data written into the staging area, while the yellow regions and checkered regions are read by two different applications.

data placement mechanism are described below.

### 6.4.1 Identifying Key Attributes of Data Read Pattern

It has been shown in previous research [54] that the data access in scientific application workflows has well-defined domain specific access patterns, which can be leveraged to increase I/O performance, especially for complex simulation workflows involving coupled codes. The key idea underlying this data placement strategy is to similarly leverage these spatial and temporal data read patterns in coupled simulation workflows. With the help from application scientists, the key spatial and temporal attributes that influence data reading performance have been identified. A *Temporal attribute* indicates when and how frequently an application will read data, e.g., the LES code reads data generated by all DNS time steps. A *Spatial attribute* defines regions within the application data domain where data is accessed.

### 6.4.2 Specifying Hints

Users can specify the temporal and spatial data access attributes identified above in the form of *hints* as follows:

$access\_hint\{T, S, N\}$

$T = \{[TS_{start}, frequency, TS_{end}]\}$;

$S = \{[Coordinates_{lower}, Coordinates_{upper}], T, Ndim\}$;

A temporal hint uses a start time step $TS_{start}$ and an end time step $TS_{end}$ to define the time interval during which data is accessed, as well as the data access frequency within this interval. Similarly, the spatial hint specifies the bounding boxes for the region of data access using boundary coordinates and the number of dimensions. $N$ indicates how many times this data object will be read. This formulation can represent most of the regular data read patterns in real scientific applications. Besides, multiple hints can be defined to enable the specification of more complex read patterns. For example, the following hints specify a data access pattern that reads data four times in the region $\{(0,0),(1,1)\}$ for every two time steps in the time interval 10 to 20, and in the region $\{(2,2),(3,3)\}$ for every time step in the time interval 40 to 45, both for a 2D data domain:

$access\_hint\{T_1, S_1, 4\} \cup access\_hint\{T_2, S_2, 4\}$ :

$T_1 = \{[10, 2, 20]\}$, $S_1 = \{[(0,0),(1,1)], T_1, 2\}$;

$T_2 = \{[40, 1, 45]\}$, $S_2 = \{[(2,2),(3,3)], T_2, 2\}$.

These hints can be updated at runtime as the simulation evolves, allowing data placement for irregular and dynamically changing data read patterns. The specific application prgoramming interfaces for initializing and updating user hints are listed in Table 6.1. Note that while the research work in this thesis has focused on data domains represented using Cartesian coordinates, this approach can be extended to other coordinate systems.

### 6.4.3 Adaptive Data Read Pattern Prediction

When user-provided hints are found to be inaccurate or are not provided, the data placement mechanism uses an adaptive data read pattern predictor. The predictor utilizes both *application-level data locality* and data reading history. *Application-level data locality* is

similar to the conventional notion of data locality but at a different granularity, and implies that if data objects for certain regions of the data domain are generated at the current time step and read by a component of the application workflow, then data objects in the same and nearby data regions that are generated during the next few time steps will likely be read by the same component. For example, the checkered data objects at time step 1 is read by one application in Figure 6.4, and consequently it is assumed that data objects in regions $\{(2,0),(3,1)\}$ will likely be read for time step 2, 3, and 4. *Application-level data locality* relies on the fact that data generated in nearby regions and time steps will probably contain related data objects, which is true for many scientific simulations.

However, *application-level temporal data locality* may not work in some cases, for example, for slow time-varying simulations where scientists are likely more interested in the phenomena across longer time intervals rather than shorter ones. Also, in other cases, such as importance-driven visualizations [83], data values impact the decision of which data objects to read, which is not fully captured by the application-level temporal data locality attribute. Therefore, the predictor tracks the history of temporal data read patterns starting from the beginning of the workflow and uses them to predict future ones following a simple prediction algorithm that is low overhead and sufficiently accurate. Specifically, this algorithm tracks and counts data read accesses at every two continuous time steps, fills up a state table, and then calculates the probabilities for all transitions. Once these probabilities have been calculated, the data read result at the current time step can be used to predict next and future ones. Table 6.2 shows an example prediction table that tracks the history of data read accesses for a data object for the previous 15 time steps (i.e., 010011010100110, where 1 represents an access and 0 represents no access) and predicts the data reading decision at the 16th. Since the data generated at the 15th time step was not read by this application, according to the state transition possibilities, it could predict that data generated at the 16th time step will not be read either. If the transition possibility is 50%, it will be predicted to read. The predicted number of times a data object will be accessed simply follows the last history record. At the beginning of a simulation, the size of the data generated is still small enough that it can be accommodated in DRAM storage within the staging area, which means that the read access information acquired during this

| *hint_init(*hint, app)* | Application *app* initializes *hint* |
|---|---|
| *hint_update(*hint, TS, app)* | Application *app* updates *hint* from time step *TS* |

Table 6.1: Programming interfaces for initializing and updating user hints.

| Current TS | Next TS | Access History | Probability |
|---|---|---|---|
| 0 | 0 | 5 | 71.4% |
| 0 | 1 | 2 | 28.6% |
| 1 | 0 | 2 | 33.3% |
| 1 | 1 | 4 | 66.6% |

Table 6.2: Example of a temporal data read pattern prediction table. The data read pattern history for the first 15 time steps is 010011010100110 (1/0: data generated at this time step is/isn't read by a specific workflow component), the data generated at the 16th time step will be predicted as 0 (i.e., not read) due to a transition probability of 71.4%.

time can serve as a good initialization of access history for the predictor.

### 6.4.4 Data Utility Management

The final objective of predicting data read pattern is to quantify the utility value of data objects, which is then used for data placement and management across different memory hierarchy levels. If a data object generated at a certain time step has been predicted to be read by an application, its *utility value* increases by the predicted number of times it will be accessed, and if a data object has been read by an application, its *utility value* decreases accordingly. When a data object is written to the staging area with sufficient available memory, it will be kept in DRAM irrespective of what its data utility value is. However, if DRAM space is constrained, data objects with higher data utility values will replace existing data objects with lower data utility values in DRAM memory, and these replaced objects will be placed directly into SSD storage.

This approach combines static and dynamic optimizations – user hints enable data placement based on static data read patterns, while the adaptive data read pattern predictor handles dynamic data read patterns. The effectiveness of this approach is experimentally evaluated in the next section.

## 6.5 Experimental Evaluation

This section presents the experimental results of adaptive application-aware data placement mechanism by using synthetic codes under different data read patterns. It also presents the performance evaluation of this multi-tiered staging approach as utilized by two applications: combustion DNS-LES simulation and plasma fusion XGC1-XGCa simulation.

The experiments were performed on ORNL's Sith InfiniBand system and Titan Cray XK7 system. Sith is an Opteron-based InfiniBand cluster equipped with SSD storage. The Sith cluster contains 40 compute nodes, each node has 4 2.3GHz 8-core AMD Opteron processors, 64GB of memory, and three 512GB Samsung 840 Pro SSDs. Titan has 18,688 compute nodes connected through a Gemini interconnect, each node has a single 16-core AMD 6200 series Opteron processor and 32GB of memory. The total system memory is 600 Terabytes. Since Titan has not been equipped with SSDs, an simple emulator is used, which utilized part of the available DRAM to mimic data writing/reading behaviors and introduced artificial delays to closer emulate real-life SSD performance.

### 6.5.1 Impact of Application-aware Data Placement

This experiment evaluates the data reading performance of this multi-tiered staging approach with internal adaptive application-aware data placement mechanism in scenarios with different data read patterns. As stated in previous sections, the data read access rate in coupled simulation workflows will impact the overall end-to-end time and performance of data-to-insight discovery. To better understand the performance and effectiveness of this approach, the experiment selected five test cases with typical data read patterns from real scientific simulation workflows. Table 6.4 summarizes these test cases and their characteristics. These tests were implemented and performed with three other data placement mechanisms in the staging area: DRAM-only placement (place data in DRAM all the time), SSD-only placement (place data in SSD all the time, no data prefetching), and application-level data locality based placement (predict and prefetch data from SSD to DRAM based on application-level temporal and spatial localities without user's hints). To avoid the influence of OS's data caching, the block cache was explicitly flushed during the experiments.

| | |
|---|---|
| Number of cores | 608 |
| No. of parallel writer cores | $8 \times 8 \times 8 = 512$ |
| No. of staging cores | 32 |
| No. of parallel reader cores | 64 |
| Volume size | $1024 \times 1024 \times 512$ |
| In staging data size (20 TSs) | 80 GB |

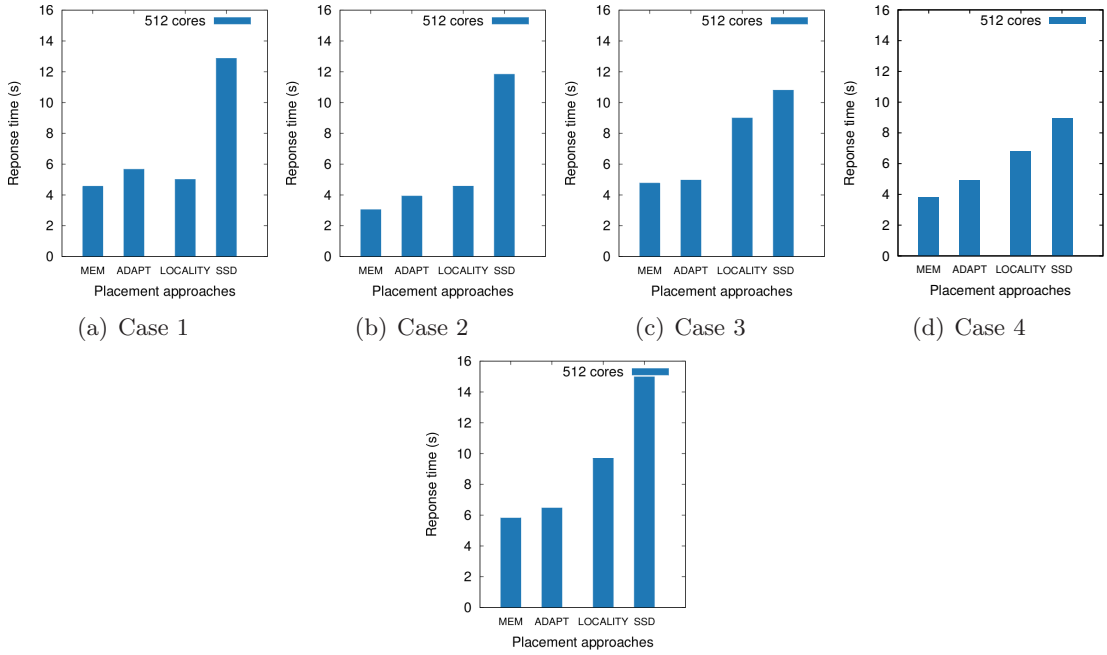
Table 6.3: Configuration of core-allocations, data domain information, and size of data cached in staging area for data placement tests using synthetic codes.

| Case # | # of readers | Data domain | Read frequency | Analytics techniques |
|---|---|---|---|---|
| 1 | One | Whole | Every single time step | Visualization[60] |
| 2 | One | Whole | Twenty consecutive time steps out of thirty time steps | Feature tracking[38], data trajectories visualization[86] |
| 3 | One | Partial | Every ten time steps | Interactive visualization, descriptive statistics[70] |
| 4 | one | Partial | Irregular temporal access pattern | importance-driven visualization[83] |
| 5 | Two | Whole | Every ten time steps for one, five time steps for the other | Visualization and topology analysis simultaneously[8] |

Table 6.4: This table summaries the characteristics of five test cases using synthetic codes .

Two synthetic application codes were used to emulate simple and generic end-to-end data movement behavior in real coupled simulation workflows: one to write data of varying sizes into the staging area and the other to read the data from the staging area. The shared computation domain in test cases is a 3-dimensional Cartesian grid, wherein each application assigns a different number of processors for each dimension, i.e., $X \times Y \times Z$. Each data element caches one `double` type variable, the location of which can be represented by using its geometric domain information, e.g., $(x, y, z)$.

In these five cases, the data for the entire domain over all of the simulation time steps are written into the staging area, but the data read pattern varies. For each case, the average data read access time over all 300 time steps was measured. The setup for these experiments is described in Table 6.3. At most, 20 time steps worth of output data is cached in the staging area at the same time. The experimental results are demonstrated in Fig. 6.5, followed by a detailed discussion and analysis of each.




(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

(e) Case 5

Figure 6.5: Comparison of average data reading response time amongst four different data placement mechanisms in five test cases with different reading patterns. MEM: DRAM-only placement (place data in DRAM all the time); SSD: SSD-only placement (place data in SSD all the time, no data prefetching); LOCALITY: application-level data locality based placement (predict and prefetch data from SSD to DRAM based on application-level data temporal and spatial localities without hints); and ADAPT: adaptive application-aware mechanism. The reading pattern used in each case is described in Table 6.4.

**Case 1 - Entire data domain for all time steps**

The data of the entire domain is read for every simulation time step. Many analysis codes, e.g. visualizations[60], use this type of data read pattern for full data resolution analysis, because it allows users to gain insights into all the physical, scientific information that is available in the output data. Obviously, when all the data is cached in DRAM, the results show the smallest average data reading response time. In contrast, the results for the SSD-only method showed the worst read access performance, because of the time cost incurred by moving data from the SSD to DRAM on demand during every data read request. The adaptive application-aware mechanism can predict and prefetch the data before each read request occurs; therefore, their performance is better than the SSD-only approach and worse than fully caching data into DRAM, due to the execution overhead. Furthermore, the adaptive mechanism in this research involves more operations after each data read, which causes a slight increase in time taken as opposed to the locality-based mechanism.

**Case 2 - Entire data domain for multiple consecutive time steps**

In analyses such as feature tracking[38] or data trajectories visualization[86], data generated over the course of multiple, consecutive time steps is needed. In this case, data reading occurs at the first consecutive 20 time steps of every 30 time steps. As the figure shows, both the application-level data locality based mechanism and the mechanism proposed here show good performance for data reading, with the latter being slightly better. The data prefetching works most of time while performing both mechanisms. Only non-consecutive data reads may cause a prefetching miss when using the application-level data locality based mechanism.

**Case 3 - Subset data domain for every 10 time steps**

In this case, data of a subset of a particular domain is read at a constant frequency, in this case, every 10 time steps. This data read pattern is usually observed in analytics that require coarse temporal and/or spatial data resolution, such as interactive visualizations and descriptive statistic analysis[70]. The application-level data locality based mechanism shows much worse performance than application-aware mechanism does. One reason for

this is, in the locality based mechanism, the data to be read is out of the temporal locality range, resulting in a prefetching miss. In contrast, the user hint provided to mechanism has already provided the data read pattern information to the middleware, so the data objects matching the hint will be prefetched from the lower memory level to a higher one. For example, after the data of time step 10 has been read, the mechanism will prefetch the data of time step 20 directly because of the user hint, but the application-level data locality based mechanism would only prefetch the data from time steps 11-15 to the memory. Since the data being read is a small subset of the original domain, the DRAM can even be leveraged to cache multiple subsets whose future access is either likely or predefined by user hints. With the help of such hints, an application-aware mechanism can more effectively utilize the limited DRAM capacity by caching as many target data sets as it deems soon-to-be accessed, resulting in an overall better data read performance.

**Case 4 - Subset data domain with irregular temporal access pattern**

This case differs from the previous case in that an irregular temporal access pattern is used. For data generated in each time step, it randomly decides whether to read it or not. This pattern may happen in a few cases, e.g., importance-driven visualization[83], wherein the coupled analytic will dynamically make the decision on the data fetching according to predefined rules. In this case, user hints are less helpful. However, with the help of the data read pattern prediction algorithm, the mechanism shows a performance advantage over the application-level data locality mechanism on overall average data read time.

**Case 5 - Entire data domain for two readers with different reading frequencies**

To reduce integration efforts required by the user and minimize performance impacts on the simulation, multiple analytics tools may be coupled with the simulation, as demonstrated in work[8, 39]. This case represents this scenario by using two readers to fetch data of an entire domain at different reading frequencies. Reader 1 fetched data every 10 time steps, while Reader 2 fetched data every 5 time steps. Data prefetching misses occurred in the locality-based mechanism when Reader 2 tries to read data out of the temporal locality range. For example, after the data of time step 20 is read by Reader 1, the locality-based

mechanism will prefetch data of time steps 20-25 to DRAM. However, the Reader 2 has not fetched data of time step 15 yet, which causes the miss penalty. In contrast, the reference number can show that data of time step 15 will be read again soon and the mechanism can guarantee the data objects to remain in DRAM until Reader 2 fetches them. Therefore, the data read performance of this application-aware mechanism is much better than that of the locality-based mechanism. From this data, it can infer that, as more readers with varying data reading frequencies are required of a workflow, this solution will provide much better data reading performance than the other methods.

### 6.5.2 Performance Evaluation with Application Drivers

Besides the test cases with synthetic codes, the experiment integrated the staging approach with two application drivers - combustion DNS-LES coupling case and plasma fusion XGC1-XGCa coupling case, and then tested its performance on Sith cluster. The objective of these real case studies is to validate the performance of the approach in real scientific data intensive simulation environment.

#### DNS-LES coupling application

As described in Section II, the one way data exchange happened from DNS to LES six times every time steps. To help the lock-step behavior recovered from the out of sync situation caused by uncertainties, 20 time steps of simulation output data will be cached in multi-tiered staging area. For comparison purpose, it performs BP-file [52] based staging, DataSpaces in-memory staging, and multi-tiered staging to temporarily cache the data in this application case, and measured the cumulative time of data reading over all 120 time steps. Table 6.5 presents the detailed experimental configuration.

Figure 6.6 demonstrates the comparison result of cumulative reading time over all of the 120 time steps by using three types of staging with DNS-LES application. At each scale, result of BP-file based staging always shows the worst performance due to the heavy I/O overhead and the DRAM based staging is the best, obviously. The multi-tiered staging shows around 31.5%, 15.2%, and 9.8% performance benefit over BP file based staging at different scales, and 23.3%, 4.5% and 5.6% performance lose over in-memory method.

| Number of cores | 130 | 260 | 520 |
|---|---|---|---|
| No. of DNS cores | $4 \times 4 \times 4 = 64$ | $8 \times 4 \times 4 = 128$ | $8 \times 8 \times 4 = 256$ |
| No. of staging cores | 4 | 8 | 16 |
| No. of LES cores | 64 | 128 | 256 |
| Volume size | $320 \times 320 \times 320$ | $320 \times 320 \times 320$ | $320 \times 320 \times 320$ |
| No. of variables | 5 | 5 | 5 |
| In staging data size (20 TSs) | 146.5 GB | 146.5 GB | 146.5 GB |

Table 6.5: Experiment configuration for three test scenarios: 130, 260, and 520 cores. It cached data generated from 20 DNS simulation Time Steps in staging for DNS-LES lock-step recovery.

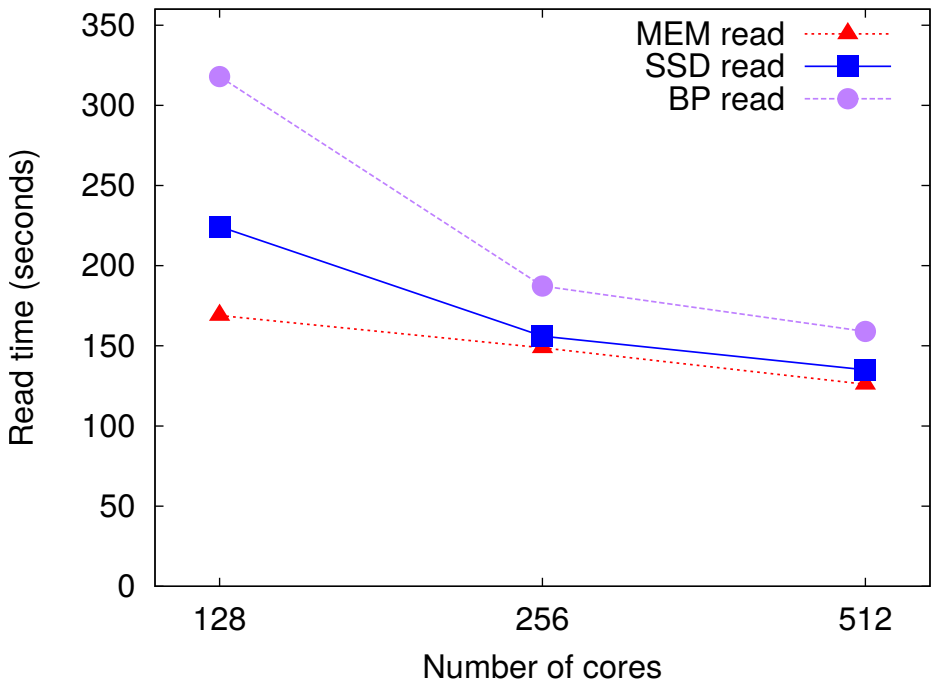



Figure 6.6: Comparison of cumulative data reading response time of reading data using BP file, SSD-based multi-tier, and in-memory staging in s3d DNS-LES coupling application.

Moreover, as the increasing number of cores are used in the experiments, the performance of multi-tiered staging method approaches that of in-memory staging method. There are two primary reasons for this phenomenon: 1) the size of the data object decreases with the increasing number of simulation cores, which reduces the penalty for each data prefetching miss accordingly; and meanwhile 2) the increasing number of smaller data objects will help application-aware data placement mechanism achieve higher hit rate of data prefetching by flexibly fitting target data objects in DRAM – as evaluated in the previous experiment section.

**XGC1-XGCa coupling application**

XGC1-XGCa coupling workflow consists of multiple coupling steps with double-way data exchange, as presented in Section II. For testing purpose, it performed two XGC1-XGCa coupling circles, each of which contains only 10 instead of thousands of time steps – because the particle data in the current coupling scheme dominates anyway. In addition, the number of particles per process was set to 1.5 million. The other experiment configuration is showed in Table 6.6. Same as the previous experiment, this test integrated and compared BP-file based staging, DataSpaces in-memory staging, and multi-tiered staging approach in this experiment.

Figure 6.7 shows the timing breakdown of the cumulative data reading time at each test case. As expected, the overall reading performance of this approach is between that of file-based staging and in-memory staging, which is the same as what is observed in previous experiments. However, may find some interesting results on these breakdowns. While the approach shows almost as good performance result as in-memory staging does for turbulence data reading, it does not show too much advantages on particle data reading. This is resulted from the imbalanced data coupling behavior in XGC1-XGCa application. Due to the small size, turbulence data could be cached in DRAM level of staging area, which will be kept there until it is retrieved. In contrast, after large size particle data is inserted, DRAM is quickly filled up and some of particle data objects have to been stored into SSD. When the data placement mechanism is performed before the coming retrieval request of entire particle data, the limited DRAM capability cannot cache all the particle data object

| Number of cores | 528 | 792 | 1056 |
|---|---|---|---|
| No. of XGC1 cores | 256 | 384 | 512 |
| No. of staging cores | 16 | 24 | 32 |
| No. of XGCa cores | 256 | 384 | 512 |
| In staging data size per coupling circle | $\sim$ 30 GB | $\sim$ 45 GB | $\sim$ 60 GB |

Table 6.6: Experiment configuration fo three test scenarios: 528, 792, and 1056 cores. It cached all the data generated from each coupling circle for data exchange between XGC1 and XGCa.

| Number of cores | 2112 | 4224 | 8448 |
|---|---|---|---|
| No. of XGC1 cores | 1024 | 2048 | 4096 |
| No. of staging cores | 64 | 128 | 256 |
| No. of XGCa cores | 1024 | 2048 | 4096 |
| In staging data size per coupling circle | $\sim$ 120 GB | $\sim$ 240 GB | $\sim$ 480 GB |

Table 6.7: This table contains the configuration of core-allocations and size of data cached in staging area for 2K, 4K, and 8K emulation test cases.

at the same time, which forces the data prefetching miss to happen across different particle data objects and delays the overall data reading.

Based on these experimental results, it is also notable that this multi-tiered staging approach can support runtime data processing and code coupling for data intensive simulation by only allocating less than extra 5% of the simulation cores on staging area.

### 6.5.3 Large Scale Emulation with XGC1-XGCa Application

This multi-tiered data staging approach has also been performed and tested at large scale with plasma fusion XGC1-XGCa application on Titan. Meanwhile, in-memory staging has been deployed and measured as well, whose performance will used as a baseline. Since there is no SSD devices on Titan, a simple emulator is built based on the performance profiling results of SSD on Sith, and then used DRAM for storage with this emulator to emulate the data writing/reading behaviors and performance of SSDs by introducing artificial delays. Before using the emulator in Titan, its functionality and performance has been validated with testing data on Sith.

The I/O performance of SSD is relevant to many factors: device manufacture, file system,

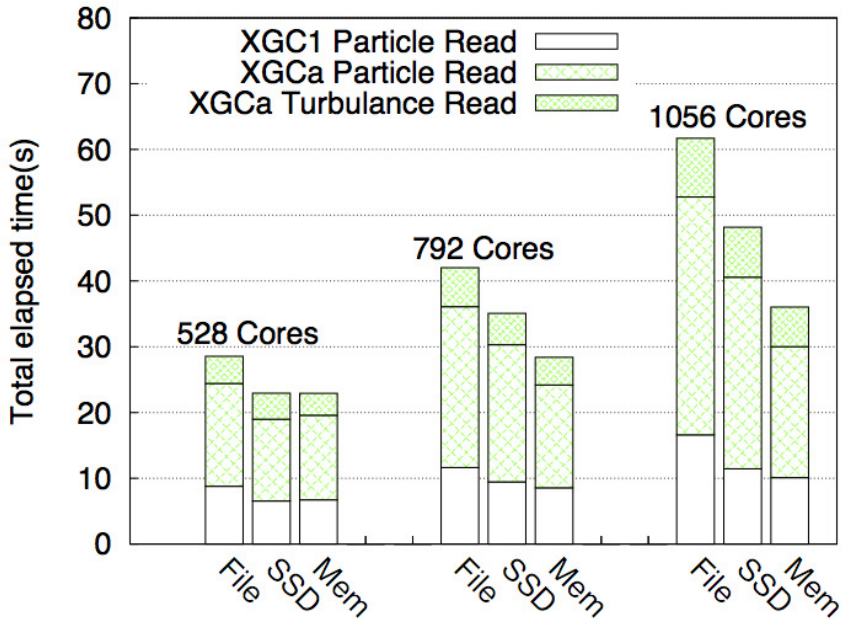



Figure 6.7: Comparison of timing breakdown of the cumulative data reading time in XGC1-XGCa coupling application.

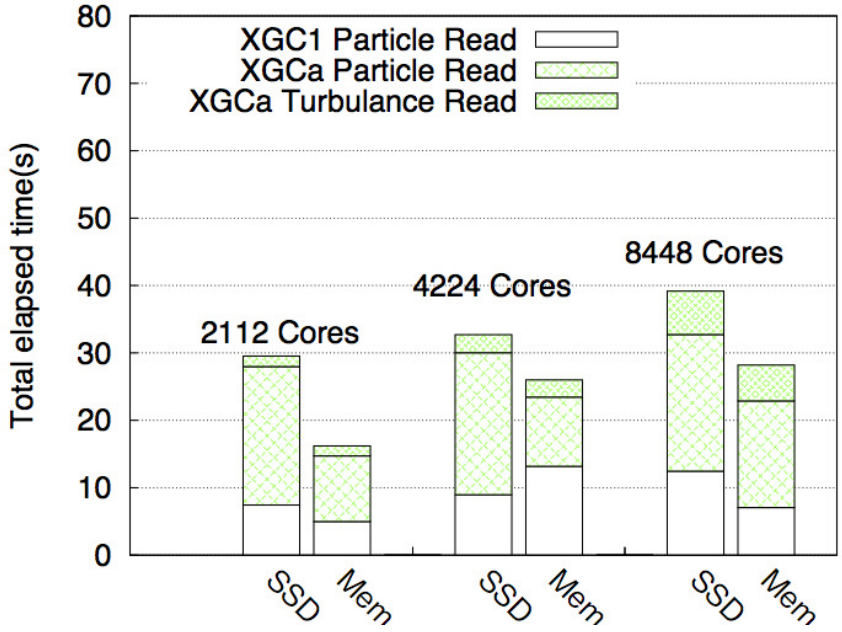



Figure 6.8: Comparison of timing breakdown of the cumulative data reading time in large scale testing with XGC1-XGCa coupling application on Titan cluster.

SSD controller, device deployment and etc. To build this simple emulator, a two-week profiling test was performed to measure the average time of copying 4KB page size data in memory and those of writing/reading same page size of data to/from SSD attached in a single node on Sith cluster. To match the deployment of CPU cores on Titan, the same deployment (16 CPU cores and 1 SSD per node) is used in this profiling test. After truncating the error results and statistically processing the profiling data within the worst 20% performance, data writing performance and data reading performance were evaluated respectively using 12 times and 4 times the cost of in-memory copy over the same data size. It is worth noting that the objective of this large scale experiments is to test the ability of this approach to respond to concurrent data reading at large scales and qualitatively check its overhead.

Figure 6.8 illustrates the experimental results while performing multi-tiered staging approach with XGC1-XGCa coupling application. Compared to the results of small scale experiments, it shows better performance on particle data reading and turbulence data reading at different scales. In addition, it shows good overall scalability and acceptable overhead with the number of application processors and data size, compared to the baseline in-memory staging approach. It can induce that the real performance of runtime would be even better than the emulation result since the performance ratios selected for the emulator are from the profiling cases with worst performance.

## 6.6 Conclusion

This chapter presented the design, implementation, and evaluation of a multi-tiered data staging runtime that leverages both DRAM and SSD to enable dynamic data staging for data-intensive coupled simulation workflows running on High End Computing systems. Underlying the runtime, it also presented the design of an adaptive application-aware data placement mechanism that dynamically manages and optimizes data placement across the DRAM and SSD storage layers using user provided hints and learned data access patterns.

In addition, the prototype of this multi-tiered data staging runtime has been deployed on the Sith InfiniBnad cluster and the Titan Cray XK7 at Oak Ridge National Lab, and been

experimentally evaluated using Combustion DNS-LES workflow and Plasma Fusion XGC1-XGCa application workflows. The experiments demonstrated that this autonomic runtime can dynamically manage and optimize data placement across deep memory hierarchy levels to efficiently support loosely coupled data intensive simulation workflows.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conlustion

Emerging coupled scientific simulation workflows running at extreme scales on leadership-class High End Computing (HEC) systems are composed of multiple applications that need exchange and share data at runtime. As system scales and application complexity grow, there are very large amounts of data that must be exchanged and shared across different coupled applications for scientific discovery and/or high fidelity modeling. Due to the huge I/O overhead, traditional file based code coupling models become infeasible. Recent simulation-time data management approaches such as in-situ/in-transit data processing using in-memory data-staging have been investigated to support large scale coupled data-intensive simulation workflows. However, with the increasing complexities of coupled applications and the architecture development of the High End Computing (HEC) systems, these coupled scientific simulation workflows running on extreme-scale computing system are presenting several new challenges.

First, many of the scientific simulations are based on dynamically adaptive formulations, which exhibit dynamic runtime behaviors and result in dynamically changing data volumes and imbalanced data distributions. For example, in AMR(Adaptive Mesh Refinement) based simulations, dynamic refinements can lead to imbalanced data distribution and heterogeneous resource (memory, CPU, network bandwidth) requirements. Such dynamic runtime behaviors increase the complexity of managing and processing the simulation data onlie while satisfying the constraints on the amount of data movement and the overhead on the simulation. They also impact on the management of the staging resources and the schedule of in-situ and/or in-transit data processing under the level of analytics and resource limitations. While current in-memory data management approaches can pre-configure the

simulation in relatively simple static workflows, they becomes ineffective when application behaviors and data requirements become dynamic. Therefore, the effectiveness of runtime data management approaches for dynamic workflow depends on the efficient and dynamic mapping of workflow components, the size and distribution of the data and the resources available in-situ and in-transit, and requires adaptive configuring the staging resources based on dynamic application constraints and requirements.

Second, architectural trends indicate that emerging systems will have increasing numbers of cores per node and correspondingly decreasing amounts of DRAM memory per core as well as decreasing memory bandwidth. These trends can significantly impact the effectiveness of the data management approaches in the runtime data processing pipeline, especially their ability to support data intensive simulation workflows and/or loosely coupled simulation workflows that demand sufficient capacity for temporally caching a large volume of data. For example, in a data-intensive coupled simulation workflow[14], data generated by one simulation may have to be consumed by another simulation or by an analytics component before it can be overwritten. As data volumes increase and memory capacity and bandwidth decrease, memory constraints can quickly become a bottleneck of performing in-situ/in-transit processing. Furthermore, the coupled simulation and/or the analytics components may require the accumulation of data over multiple time steps[65], which can further accentuate this problem.

This thesis identifies and addresses the above key runtime data management requirements for coupled simulation workflows at extreme scales. Specifically, this thesis explores an autonomic data management approach to enable efficient data management at runtime in an adaptive manner that can dynamically respond to the varying data management requirements. The presented autonomic data management approach can efficiently enable the applications and the runtime system to adapt and tune their own behaviors – e.g., data quality, data placement, placement of analytics, and resource allocation – automatically by following pre-defined strategies and policies. These strategies and policies are selected adaptively based on the varying operating environment statuses and specific data management objectives. This thesis makes the following contributions:

- Formulation of an autonomic data management abstraction for coupled scientific workflows in extreme scale High End Computing (HEC) systems based on dynamic runtime data management requirements, characteristics of application behaviors, and environment of the system/resource. This provides the foundation for the conceptual framework of the approach and allows for the specification of its key components and required functionalities. This framework provides a variety of mechanisms that will be automatically orchestrated at runtime to respond to the heterogeneity and dynamics of the applications and the infrastructure.

- Design and development of effective cross-layer adaptation policies and mechanisms to enable dynamic data management in *tightly coupled* workflows containing applications exhibiting dynamic behaviors. These adaptations include (1) adaptation of the spatial resolution at which the data is processed, (2) dynamic placement and scheduling of data processing kernels, (3) dynamic allocation of in-transit resources, and (4) coordinated adaptations that dynamically combine these adaptations at the different layers automatically at runtime. This dynamic data management scheme reduces the in-network data movement, improves overall performance and increases resource efficiency at runtime.

- Design and development of a multi-tiered staging runtime that spans both DRAM and a deep memory hierarchy layer – solid state disks (SSD) – to support both dynamic data management and code coupling for *loosely coupled* data intensive simulation workflows. This multi-tiered staging runtime allows entire workflows to cache larger data volumes that may exceed available DRAM memory within the staging area for runtime data management. In addition, this staging method can be deployed in future High End Computing systems by replacing the SSD layer with another faster memory hierarchy device, such as NVRAM.

- Design and implementation of an adaptive application-aware data placement mechanism that can dynamically manage and optimize data placement across the DRAM and SSD storage layers using data access patterns and user provided hints. This

mechanism can support effective data prefetching from a lower level of memory hierarchy (i.e, SSD) to local DRAM in hybrid staging areas, reducing access overheads and improving the end-to-end performance.

- Integration of the prototype of the autonomic data management approach with real-word coupled scientific simulation workflows, e.g., combustion and fusion applications; and demonstrate its effectiveness in improving overall performance and accelerating scientific discovery.

## 7.2 Future Work

This thesis proposed an autonomic data management approach, explained its design and prototype development, and experimentally demonstrated its effectiveness in supporting dynamic runtime data managements, and in improving overall performance. This research work can be extended in several directions, as summarized below:

- **Exploring power-constrained data management**: The data management policies and mechanisms in this thesis are mostly designed and implemented from a performance perspective. One of the potential future research directions is to explore the impacts of this autonomic data management approach in terms of the energy consumption, as well as power and performance trade-offs for data intensive simulation workflows with different data management algorithms and policies. By developing relevant autonomic policies, the autonomic data management abstraction can incorporate with power/energy efficiency and power/performance trade-offs as part of the autonomic objectives. Additionally, the impact of the different deep memory hierarchy levels and different runtime data placement strategies with respect to the energy consumption can be explored.

- **Utilizing multiple memory hierarchy levels** This thesis focuses on utilizing one extra memory level, i.e., SSDs. However, there are many new fast storage technologies. For example, Cray announced "burst buffer"[2], which is an intermediate, high-speed layer of storage that is positioned between DRAM and the parallel file system (PFS). Burst buffer can firstly cache the bulk data produced by the applications at a rate

that is a hundred times higher than using PFS; and then seamlessly drains the data to PFS in the background. This burst buffer is built from NVRAM devices that have several desirable properties, such as high I/O throughput, low access latency, and higher reliability. Moreover, the planned extreme scale supercomputer "Summit"[3], which is being built by IBM and will be installed at Oak Ridge National lab in 2018, also has NVRAM devices installed as an additional memory layer. Therefore, another direction is to further expand the staging area and data processing approaches by adding intermediate storage layers between DRAM and SSD using NVRAM devices. Effective and adaptive data placement mechanisms also have to address the runtime data placement and management challenges.

- **Domain-specific programming abstractions** The third research direction is to design and formalize programming abstractions to support autonomic adaptations and dynamic data management. This abstraction must enable users to express autonomic objectives and constraints as well as to formulate autonomic policies. Besides, as described in this thesis, domain specific information can be leveraged to better support data management adaptations and optimizations in coupled simulation workflows. Therefore, programming support of effectively expressing these information is required. Last but not least, the programming abstraction should enable users to access resources of different memory hierarchy levels explicitly and to place data objects across these levels easily.

# References


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

[2] Cray burst buffer website, "http://www.hpcwire.com/2014/05/01/burst-buffers-flash-exascale-potential/".

[3] Summit website, "https://www.olcf.ornl.gov/summit/".

[4] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. Just In Time: Adding Value to The IO Pipelines of High Performance Applications with JITStaging. In *Proc. 20th International Symposium on High Performance Distributed Computing (HPDC'11)*, June 2011.

[5] 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.

[6] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for web server end-systems: a control-theoretical approach. *Parallel and Distributed Systems, IEEE Transactions on*, 13(1):80–96, Jan 2002.

[7] M. A. Amer, A. Chervenak, and W. Chen. Improving scientific workflow performance using policy based data placement. In *Proceedings of the 2012 IEEE International Symposium on Policies for Distributed Systems and Networks*, POLICY '12, pages 86–93, Washington, DC, USA, 2012. IEEE Computer Society.

[8] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. W. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, 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.

[9] T. E. Bihari and K. Schwan. Dynamic adaptation of real-time software. *ACM Trans. Comput. Syst.*, 9(2):143–174, May 1991.

[10] D. Borthakur. Hdfs architecture guide. 2008.

[11] J.-M. F. Brad Whitlock and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11)*, April 2011.

[12] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp. Parallel i/o prefetching using mpi file caching and i/o signatures. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 44:1–44:12, Piscataway, NJ, USA, 2008. IEEE Press.

[13] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorski, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using S3D. *Comp. Sci. Disc.*, 2:1–31, 2009.

[14] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorski, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science and Discovery*, 2:1–31, 2009.

[15] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[16] A. L. Chervenak, A. Sim, J. Gu, R. Schuler, and N. Hirpathak. Efficient data staging using performance-based adaptation and policy-based resource allocation. In *Proceedings of the 2014 22Nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '14, pages 244–247, Washington, DC, USA, 2014. IEEE Computer Society.

[17] A. L. Chervenak, A. Sim, J. Gu, R. E. Schuler, and N. Hirpathak. Adaptation and policy-based resource allocation for efficient bulk data transfers in high performance computing environments. In *Proceedings of the Fourth International Workshop on Network-Aware Data Management*, NDM '14, pages 1–8, Piscataway, NJ, USA, 2014. IEEE Press.

[18] A. L. Chervenak, D. E. Smith, W. Chen, and E. Deelman. Integrating policy with scientific workflow management for data-intensive applications. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, SCC '12, pages 140–149, Washington, DC, USA, 2012. IEEE Computer Society.

[19] W. D. Collins, C. M. Bitz, M. L. Blackmon, G. B. Bonan, C. S. Bretherton, J. A. Carton, P. Chang, S. C. Doney, J. J. Hack, T. B. Henderson, J. T. Kiehl, W. G. Large, D. S. McKenna, B. D. Santer, and R. D. Smith. The Community Climate System Model Version 3 (CCSM3). *Journal of Climate*, 19(11):2122–2143, 2006.

[20] E. F. D'Azevedo, J. Lang, P. H. Worley, S. A. Ethier, S.-H. Ku, and C. Chang. Hybrid mpi/openmp/gpu parallelization of xgc1 fusion simulation code. In *Supercomputing Conference 2013*, 2013.

[21] A. De, M. Gokhale, R. Gupta, and S. Swanson. Minerva: Accelerating data analysis in next-generation ssds. In *Proceedings of the 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '13, pages 9–16, 2013.

[22] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. hui Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid, 2004.

[23] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, Dec. 2006.

[24] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey. Fulfilling the vision of autonomic computing. *Computer*, 43(1):35–41, Jan 2010.

[25] 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.

[26] C. Docan, M. Parashar, and S. Klasky. DART: A Substrate for High Speed Asynchronous Data IO. In *Proc. of 17th International Symposium on High Performance Distributed Computing (HPDC'08)*, June 2008.

[27] 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.

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

[29] M. Dorier, S. Ibrahim, G. Antoniu, and R. Ross. Omnisc'io: A grammar-based approach to spatial and temporal i/o patterns prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 623–634, Piscataway, NJ, USA, 2014. IEEE Press.

[30] 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.

[31] K. Geebelen, E. Kulikowski, E. Truyen, and W. Joosen. A mvc framework for policy-based adaptation of workflow processes: A case study on confidentiality. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 401–408, July 2010.

[32] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. volume 37, Oct. 2003.

[33] C. Gniady, A. R. Butt, and Y. C. Hu. Program-counter-based pattern classification in buffer caching. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 27–27, Berkeley, CA, USA, 2004. USENIX Association.

[34] K. Gupta, R. Jain, I. Koltsidas, H. Pucha, P. Sarkar, M. Seaman, and D. Subhraveti. Gpfs-snc: An enterprise storage framework for virtual-machine clouds. *IBM Journal of Research and Development*, 2011.

[35] J. He, J. Bent, A. Torres, G. Grider, G. Gibson, C. Maltzahn, and X.-H. Sun. I/o acceleration with pattern detection. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 25–36, New York, NY, USA, 2013. ACM.

[36] 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.

[37] IBM. Autonomic computing: Ibm's perspective on the state of information technology.

[38] 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.

[39] T. Jin, F. Zhang, Q. Sun, H. Bui, M. Parashar, H. Yu, S. Klasky, N. Podhorszki, and H. Abbasi. Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, 2013.

[40] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[41] 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.

[42] T. Kosar and M. Livny. Stork: making data placement a first class citizen in the grid. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 342–349, 2004.

[43] T. M. Kroeger and D. D. E. Long. The case for efficient file access pattern modeling. In *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, HOTOS '99, pages 14–, Washington, DC, USA, 1999. IEEE Computer Society.

[44] S. Ku, C. Chang, and P. Diamond. Full-f gyrokinetic particle simulation of centrally heated global itg turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry. *Nuclear Fusion*, 49(11):115021, 2009.

[45] 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.

[46] P. Lama and X. Zhou. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 63–72, New York, NY, USA, 2012. ACM.

[47] S. Lasluisa, F. Zhang, T. Jin, I. Rodero, H. Bui, and M. Parashar. In-situ feature-based objects tracking for data-intensive scientific and enterprise analytics workflows. *Cluster Computing*, pages 1–12, 2014.

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

[49] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, 2010.

[50] H. Liu, V. Bhat, M. Parashar, and S. Klasky. An autonomic service architecture for self-managing grid applications. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, GRID '05, pages 132–139, Washington, DC, USA, 2005. IEEE Computer Society.

[51] H. Liu and M. Parashar. Accord: a programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3):341–352, May 2006.

[52] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu. Hello adios: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience*, 2013.

[53] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. Parekh. Online response time optimization of apache web server. In *Proceedings of the 11th International Conference on Quality of Service*, IWQoS'03, pages 461–478, Berlin, Heidelberg, 2003. Springer-Verlag.

[54] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: Reading patterns for extreme scale science io. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC'11, 2011.

[55] 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.

[56] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[57] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms*. *Real-Time Syst.*, 23(1/2):85–126, July 2002.

[58] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and The Lepler System: Research Articles. *Concurrency and Computation : Practical and Experience*, 18:1039–1065, August 2006.

[59] J. Luxon. A design retrospective of the diii-d tokamak. *Nuclear Fusion*, 42(5):614, 2002.

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

[61] X. Ma, J. Lee, and M. Winslett. High-level buffering for hiding periodic output cost in scientific simulations. *Parallel and Distributed Systems, IEEE Transactions on*, 17(3):193–204, March 2006.

[62] T. M. Madhyastha and D. A. Reed. Learning to classify parallel input/output access patterns. *IEEE Trans. Parallel Distrib. Syst.*, 13(8):802–813, Aug. 2002.

[63] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153, London, UK, UK, 1995. Springer-Verlag.

[64] G. Malewicz, I. Foster, A. Rosenberg, and M. Wilde. A tool for prioritizing dagman jobs and its evaluation. *Journal of Grid Computing*, 5(2):197–212, 2007.

[65] A. Mascarenhas, R. Grout, P.-T. Bremer, E. Hawkes, V. Pascucci, and J. Chen. Topological feature extraction for comparison of terascale combustion simulation data. In V. Pascucci, X. Tricoche, H. Hagen, and J. Tierny, editors, *Topological Methods in Data Analysis and Visualization*, Mathematics and Visualization, pages 229–240. Springer Berlin Heidelberg, 2011.

[66] 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.

[67] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May 1999.

[68] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In *Proceedings of the 20th International Conference on Software Engineering*, ICSE '98, pages 177–186, Washington, DC, USA, 1998. IEEE Computer Society.

[69] M. Parashar and S. Hariri. Autonomic computing: An overview. In *Proceedings of the 2004 International Conference on Unconventional Programming Paradigms*, UPP'04, pages 257–269, Berlin, Heidelberg, 2005. Springer-Verlag.

[70] P. Pebay, D. Thompson, and J. Bennett. Computing contingency statistics in parallel: Design trade-offs and limiting cases. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 156–165, Sept 2010.

[71] R. Prabhakar, S. Vazhkudai, Y. Kim, A. Butt, M. Li, and M. Kandemir. Provisioning a multi-tiered data staging area for extreme-scale machines. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 1–12, June 2011.

[72] 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.

[73] S. Sadjadi and P. McKinley. Act: an adaptive corba template to support unanticipated adaptation. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 74–83, 2004.

[74] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware qos management in web servers. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, RTSS '03, pages 63–, Washington, DC, USA, 2003. IEEE Computer Society.

[75] G. Staffelbach, J. M. Senoner, L. Gicquel, and T. Poinsot. High performance computing for computational science - vecpar 2008. chapter Large Eddy Simulation of Combustion on Massively Parallel Machines, pages 444–464. Springer-Verlag, Berlin, Heidelberg, 2008.

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

[77] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 93–102, Sept 2011.

[78] D. Tiwari, S. Boboila, S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin. Active flash: Towards energy-efficient, in-situ data analytics on extreme-scale machines. In *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, pages 119–132, 2013.

[79] N. Tran and D. Reed. Automatic arima time series modeling for adaptive i/o prefetching. *IEEE Trans. Parallel Distrib. Syst.*, 15(4):362–377, Apr. 2004.

[80] C. Ururahy, N. Rodriguez, and R. Ierusalimschy. Alua: Flexibility for parallel programming. *Comput. Lang. Syst. Struct.*, 28(2):155–180, Dec. 2002.

[81] 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.

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

[83] C. Wang, H. Yu, and K.-L. Ma. Importance-driven time-varying data visualization. *Visualization and Computer Graphics, IEEE Transactions on*, pages 1547–1554, 2008.

[84] 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.

[85] J. Wei and C.-Z. Xu. eqos: Provisioning of client-perceived end-to-end qos guarantees in web servers. *Computers, IEEE Transactions on*, 55(12):1543–1556, Dec 2006.

[86] J. Wei, H. Yu, R. Grout, J. Chen, and K.-L. Ma. Dual space analysis of turbulent combustion particle data. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 91–98, 2011.

[87] Q. Wu and T. Zhang. Ofwar: Reducing ssd response time using on-demand fast-write-and-rewrite. *Computers, IEEE Transactions on*, pages 1–1, 2013.

[88] Y. Yin, J. Li, J. He, X.-H. Sun, and R. Thakur. Pattern-direct and layout-aware replication scheme for parallel i/o systems. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 345–356, May 2013.

[89] C. S. Yoo, R. Sankaran, and J. H. Chen. Three-dimensional direct numerical simulation of a turbulent lifted hydrogen jet ame in heated coow: Flame stabilization and structure. *Journal of Fluid Mechanics*, 640:453–481, 2009.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] X. Zhang, K. Davis, and S. Jiang. Opportunistic data-driven execution of parallel programs for efficient i/o services. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, IPDPS '12, pages 330–341, Washington, DC, USA, 2012. IEEE Computer Society.

[95] 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.

[96] 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.