AUTONOMIC MANAGEMENT OF DATA STREAMING AND IN-TRANSIT PROCESSING FOR DATA INTENSIVE SCIENTIFIC WORKFLOWS

BY VIRAJ BHAT

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

Professor Manish Parashar

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2008

ABSTRACT OF THE DISSERTATION

Autonomic Management of Data Streaming and In-Transit Processing for Data Intensive Scientific Workflows

by Viraj Bhat

Dissertation Director: Professor Manish Parashar

High-performance computing is playing an important role in science and engineering and is enabling highly accurate simulations, which provide insights into complex physical phenomena. A key challenge is managing the enormous data volumes and high data rates associated with these applications, so as to have minimal impact on the execution of the simulations. Furthermore these applications are based on seamless interactions and coupling between multiple and potentially distributed computational, data and information services. This requires addressing the natural mismatches in the ways data is represented in different workflow components and on a variety of machines, and being able to "outsource" the required data manipulation and transformation operations to less expensive commodity resources "in-transit". Satisfying these requirements is challenging, especially in large-scale and highly dynamic in-transit environments with shared computing and communication resources, resource

heterogeneity in terms of capability, capacity, and costs, and where application behaviors, needs, and performance are highly variable.

In this research we address these requirements by developing a data streaming and in-transit data manipulation framework that provides mechanisms as well as the management strategies for large scale and wide-area data intensive scientific and engineering workflows. The main objectives of this research are: (1) developing an end-to-end QoS management framework for data intensive applications so that it is able to provide robust underlying support for asynchronous, high-throughput, low-latency data streaming, and (2) effectively and opportunistically utilize resources in-transit for data processing, to match data mismatches between application entities executing in scientific workflows.

In this thesis, we address problem at two levels, the first or application level deals with satisfying QoS goals at the end points. Specifically, it ensures that the data is delivered in a timely manner, with no loss at the source or destination, and with minimal storage requirements at the end-points. The solution couples model-based limited look-ahead controllers (LLC) with rule-based managers to satisfy data streaming requirements under various operating conditions. The second or in-transit level focuses on scheduling in-transit computations and data transfer in an opportunistic manner on the in-transit overlay resources taking into account the higher level QoS goals of the source and the sink. Additionally the in-transit level management is coupled with the application level management at end points to manage QoS of grid workflows.

This research is driven by the requirements of the Fusion Simulation Project (FSP), which forms the basis of a predictive plasma edge simulation capability to support next-generation burning plasma experiments such as the International Thermonuclear Experimental Reactor (ITER). These scientific workflows require in-transit data manipulation and streaming in a wide area environment.

The self-managing data streaming service developed using this approach for the

FSP workflow minimizes streaming overheads on the executing simulation to about 2% of the simulation execution time, reduces buffer occupancy at the source and thus prevents data loss. Additionally experiments with self-managing data streaming and in-transit processing demonstrates that adaptive processing using this service during network congestions decreases average idle time per data block from 25% to 1%, thereby increasing utilization at critical times. Furthermore, coupling end-point and in-transit level management during congestion reduces average buffer occupancy at in-transit nodes from 80% to 60.8%, thereby reducing load and potential data loss.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Manish Parashar, for his invaluable guidance, support and encouragement during the course of this work and throughout my graduate studies at Rutgers University. I am thankful to Dr. Ivan Marsic, Dr. Dario Pompili (all at Rutgers University, NJ), Dr. Scott Klasky (Oak Ridge National Lab (ORNL), TN) and Dr. Christopher Marty (Bloomberg L.P., NY) for being on my thesis committee and for their advice and suggestions.

I thank Dr. Nagarajan Kandasamy (Drexel University, PA) for valuable research discussions and collaboration related to this research. I thank Dr. Micah Beck (University of Tennessee of Knoxville, TN) and Scott Atchley (Myricom Inc.) for valuable contributions towards this research. I also thank Dr. Stephane Ethier, Doug McCune and Dr. Ravi Samtaney (all at Princeton Plasma Physics Laboratory (PPPL), NJ) for collaboration on application codes.

I am grateful to the CAIP computer facilities staff James Chun and Bill Kish at Rutgers University and PPPL network staff for their assistance and support. I would like to thank my colleagues at The Applied Software Systems Laboratory (TASSL) especially Sumir Chandra, Vincent Matossian and Ciprian Docan for their cooperation and a wonderful work environment. Above all, I am thankful to my family and friends for their constant love and support.

The research presented in this thesis is supported in part by the National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354 and IIS 0430826, and by the US Department of Energy via the grant number DE-FG02-06ER54857. It was also supported by USDOE Contract no. DE-AC020-76-CH03073. This research used resources of the National

Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. I would also like to thank Tech-X Corporation for their support.

Dedication

To my family

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Motivation

1.1.1 Data Intensive Application Workflows for High Performance Computing

The emergence of high-performance distributed computational environments is enabling new practices in science and engineering. These are based on seamless interactions and couplings across multiple and potentially distributed computational, data, and information services. For example, current fusion simulation efforts are exploring coupled models and codes that simultaneously simulate separate application processes and run on different High Performance Computing (HPC) resources at supercomputing centers. These codes will need to interact, at runtime, with each other, and with additional services for online data monitoring, data analysis and visualization, and data archiving. Furthermore these analytics require high throughput data movement methods that shield scientists from machine-level details, such as the throughput achieved by a file system or the network bandwidth available to move data from the supercomputer site to remote machines on which the data is analyzed or visualized. Hence this requires a new computing environment in which scientists can ask, "What if I increase the pressure by a factor of ten," and have the analytics run the appropriate methods to examine the effects of such a change. Since high performance simulations run for long periods of times, it is possible for scientists to do in-situ visualization during the lifetime of the run. The outcome of this

approach is a paradigm shift in which potentially plentiful computational resources (e.g., multi-core and accelerator technologies) are used to replace potentially scarce I/O capabilities by, for instance, introducing high performance I/O with visualization without *polluting* the simulation code with additional visualization routines.

Such "analytic I/O" operations require efficient movement of data from compute nodes of supercomputers to locations such as GPGPU's (General-Purpose computing on Graphics Processing Units) or commodity clusters where data manipulation through analysis and visualization is performed and/or to other I/O nodes where data is archived to disk. Furthermore, the locations where analytics are performed are flexible, with simple filtering or data reduction actions able to run on clusters, data routing or reorganization performed on I/O nodes, and more generally, with metadata generation (i.e., the generation of information about data) performed wherever appropriate to match end user requirements. For instance, analytics may require that certain data be identified and tagged on I/O nodes while it is being moved, so that it can be routed to analysis or visualization machines. At the same time, for performance and scalability, other data may be moved to disk in its raw form, to be reorganized later into file organizations desired by end users. In all such cases, however, high throughput data movement is inexorably tied to data analysis, annotation, and cataloging, thereby enhancing raw data to become the information required by end users.

1.1.2 Driving Application

The DoE SciDAC CPES Fusion Simulation Project(FSP) [46] aims to develop a new integrated Grid-based predictive plasma edge simulation capability to support next-generation burning plasma experiments such as the International Thermonuclear Experimental Reactor (ITER). One of driving application for the FSP and this thesis is the GTC [52] fusion simulation that scientists execute on the 250+ Tflop computer at Oak Ridge National Laboratory (ORNL). GTC is a state of the art global fusion

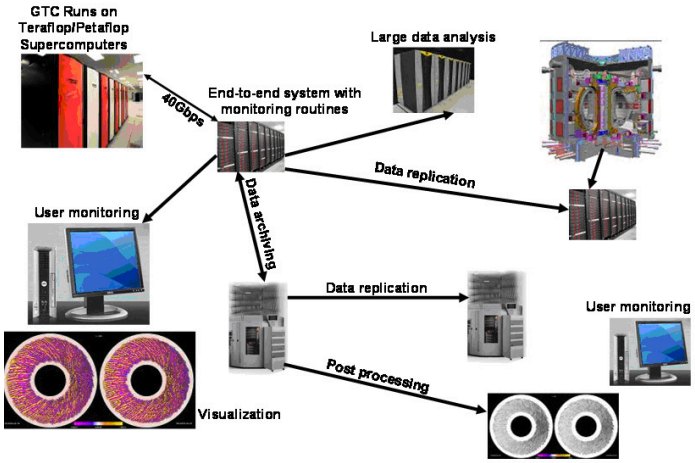

Figure 1.1: Workflow for the Fusion Simulation Project

code that has been optimized to achieve high efficiency on a single computing node and nearly perfect scalability on massively parallel computers. It uses the Particle-In-Cell (PIC) technique to model the behavior of particles and electromagnetic waves in a toroidal plasma in which ions and electrons are confined by intense magnetic fields. One of the goals of GTC simulation is to resolve the critical question of ρ^* scaling of confinement in large tokamaks such as ITER. The scientific impact of these simulations will be substantial, as it will further the understanding of Collisionless Trapped Electron Mode (CTEM) turbulence by validating this against modulated Electron Cyclotron Heating heat pulse propagation in current fusion reactors.

In order to understand these effects, and validate the simulations against experiments, scientists need to record enormous amounts of data. Further it is expected that a number of simulations are to be executed for the scientific discovery process, each simulation running on roughly thirty-two thousand processors, and occupying over 50TB of memory on the Cray XT computer at ORNL. Ideally, it would be desired to store all of the particle information generated by the simulations, but this will be a daunting task, as it requires to store PB(PetaBytes) of information to the HPSS with a sustained throughput of over 300 GB/sec while maintaining a low overhead on the simulation. Finally, since human and system errors can occur, it is critical to

monitor the simulation during their execution via in-transit functions such as multi-dimensional FFTs, correlation functions over a specified time range or coupled codes executing on cheaper resources. These techniques can save over 100K CPU hours for every hour of the simulation run on machines such as the Cray XT at ORNL.

Figure 1.1 shows a typical workflow comprising of coupled simulation codes the edge turbulence particle-in-cell (PIC) code, Gyrokinetic Toroidal Code (GTC) [52] and the microscopic MHD (Magnetohydrodynamics) code (Multilevel 3D or M3D) [25] running simultaneously on thousands of processors at various supercomputing centers. As shown in the figure data produced by these simulations must be streamed live to remote sites for online simulation monitoring and control, simulation coupling, data analysis and visualization, online validation, and archiving.

1.2 Problem Description

Scientific application workflows require robust underlying support for asynchronous, high throughput, low-latency data streaming between interacting components. For example in a fusion workflow, for instance, large volumes and heterogeneous types of data generated have to be continuously streamed from a petascale machine's compute to its I/O partition, and from there to compute systems running coupled simulation components and to auxiliary data analysis and storage machines. Requirements imposed on such data-streaming and in-transit processing service are that it must

1. It should have minimal impact on the execution of the petascale simulations with less than 10% of the simulation time spent on data transfer activities

2. It should satisfy stringent application/user space and time constraints by ensuring "in-time" delivery of data at end-points

3. It should deal with natural mismatches in the ways data is represented in different components and on different machines while ensuring the strict deadline

requirements

4. It should also guarantee that no data is lost in-transit processing or during data transfer

Satisfying the above requirements in large-scale, heterogeneous and highly dynamic Grid environments with shared computing and communication resources, and where the application behaviour and performance is highly variable, is a significant challenge. It typically involves multiple functional and performance-related parameters that must be dynamically tuned to match the prevailing application requirements and Grid operating conditions. As scientific applications grow in scale and complexity, and with many of these applications running in batch mode with limited or no runtime access, maintaining desired QoS using current approaches based on ad hoc manual tuning and heuristics is not just tedious and error-prone, but infeasible. A practical data streaming service and in-transit processing service must, therefore, be largely *self-managing*, i.e., it must dynamically detect and respond, quickly and correctly, to changes in application behaviour and state of the underlying resources.

1.2.1 Requirements for Data Streaming

The fundamental requirement of the wide area data streaming service is to efficiently and robustly stream data from live simulations to remote services while satisfying the following constraints: (1) Enable high-throughput, low-latency data transfer to support near real-time access to the data. (2) Minimizing overheads on the executing simulation. The simulation executes in batch for days and we would like the overhead of the streaming on the simulation to be less than 10% of the simulation execution time. (3) Adapting to network conditions to maintain desired QoS. The network is a shared resource and the usage patterns typically vary constantly. (4) Handle network failures while eliminating loss of data. Network failures usually lead to buffer overflows, and data has to be written to local disks to avoid loss. This increases

the overhead in the simulation. Further, the data is no longer available for remote analysis.

1.2.2 Requirements for In-Transit Processing

In-transit processing has to deal with multiple QoS requirements consisting of possibly unknown number of senders or initial data producers and sinks or final data receivers. The main requirements for data in-transit processing service are that it should (1) Ensure "in-time" delivery of data while (2) Introducing low overhead on-transit functions and enabling quick forwarding of simulation data at the in-transit nodes (3) Opportunistically process data so that the best quality of processed data reaches the sink "in-time" (4) Effectively schedule and manage in-transit processing while satisfying the above requirements - this is particularly challenging due to the limited capabilities and resources and the dynamic capacities of the typically shared processing nodes.

1.3 Problem Statement

The goal of the proposed research is to develop, deploy, and evaluate a self-managing framework for high-performance and robust data transport, "in-transit" data manipulation, and online analysis for data-intensive scientific applications. The objectives of such a framework are to provide:

- *Asynchronous data capture* and I/O services for capturing and then transporting the right simulation data at the right time ("in-time") from the computational nodes of leadership class computing platforms to its I/O nodes and onwards. The primary objectives of these services are to minimize data capture and I/O costs, their impact on the performance of the simulation, and to provide end users with the data needed to monitor application progress and/or control them.

One reason for the latter is to quickly catch interesting simulation behaviors (including problematic ones).

- *High-throughput, low-latency data streaming* services will enable the streaming of terabytes of simulation data between coupled simulation components as well as between simulations and online analysis components, while satisfying space and time constraints.

- *In-transit data manipulation* services will enable efficient application-specific manipulations of data as it is being captured, moved, analyzed, and stored, to enable near real-time analysis and visualization capabilities and perhaps more importantly, to leverage the computational capabilities of future machines to better deal with the data floods expected from petascale simulations.

- *Control driven Policy-based mechanisms for QoS management* will enable data streaming services to be largely self-managing, enabling them to dynamically detect and respond, quickly and correctly, yet compliant with end user policies, to changes in application behavior, needs, and underlying system state. Such automation is essential to be able to scale these services to future petascale machines and to the support environments (e.g., auxiliary analysis, storage, and visualization clusters) in which they operate.

- *Validate* the performance of such a framework on real scientific application workflows such as the FSP executing on distributed heterogeneous resources having varied QoS requirements.

1.4 Overview and Research Approach

To address the problem of self-managing data streaming and in-transit processing we intend build a two level self-managing framework as illustrated in Figure 1.2 to address QoS issues in data intensive scientific workflows operating on the Grid.

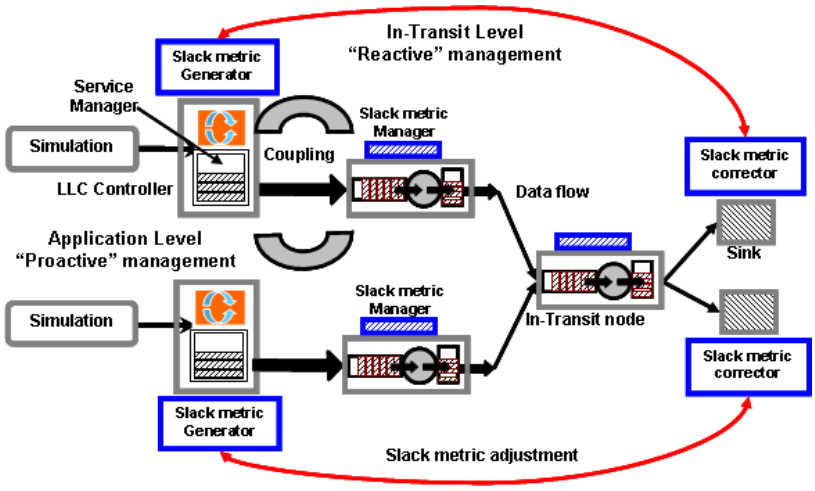

Figure 1.2: Two Level Self-Managing Cooperative Framework for Data Intensive Scientific Workflows involving Data in-Transit

A conceptual overview of the self-managing data streaming and in-transit processing service for Grid-based data intensive scientific workflows is presented in Figure 1.2. It consists of two key components: The first is an application level data streaming service, which provides adaptive buffer management mechanisms and proactive QoS management strategies based on online control and user-defined polices, at application end-points. The application level component also captures constraints for in-transit processing using a "slack metric" generated at the application level. The second component provides scheduling mechanisms and adaptive runtime management strategies for in-transit data manipulation and transformation. These two components work cooperatively to address the overall application constraints and QoS requirements.

1.4.1 Application Level Self-Managing Data Streaming

The application level self-managing data streaming service combines model-based limited look-ahead controllers (LLC) and rule-based autonomic managers with adaptive multi-threaded buffer management and data transport mechanisms at the application endpoints. It is constructed using the Accord-WS infrastructure for self-managing Grid services and supports high throughput, low latency, robust application level

data streaming in wide-area Grid environments. The autonomic data streaming service consists of a service manager coupled with an LLC controller. The service manager monitors the state of the service and its execution context, collects and reports runtime information, and enforces the adaptation actions determined by its controller. Augmenting the element manager with an LLC controller allows human defined adaptation polices, which may be error-prone and incomplete, with mathematically sound models and optimization techniques for more robust self-management. Specifically, the controller decides when and how to adapt the application behavior and the service managers focus on enforcing these adaptations in a consistent and efficient manner. Additionally the LLC controller is enhanced with a slack metric generator which fixes a time deadline for delivery of simulation data at the sink. The input from the LLC controller, sink slack correctors and in-transit slack managers, provide valuable updates to the slack generator at the application level.

1.4.2 In-Transit Level Data Processing

The in-transit data manipulation framework consists of a dynamic overlay of available in-transit processing nodes (e.g., workstations or small to medium clusters) with heterogeneous capabilities and loads. Note that these nodes may be shared across multiple workflows. Each node can perform a limited number of operations on the in-transit data, these include processing, buffering and forwarding. The slack metric managers at the in-transit level update the slack metric generated at the application level after each of the following operations on the data. The processing on the in-transit data depends on the capacity and capability of the node, the amount of processing that is still required on the data which is depicted as the slack metric (generated at the application level using the slack metric generator), the network conditions between the in-transit nodes and the load on a particular in-transit node. The amount of processing completed is logged in the data block itself. The goal of the in-transit processing is to process as much data as possible before the data reaches

the sink. A processing that is not completed in-transit will have to be performed at the sink. The current design of the framework assumes that each node can perform any of the required data manipulations functions. It also assumes that the in-transit functions do not change the size of the data items during in-transit.

1.4.3 Cooperative Management

The application level and in-transit management are coupled to achieve cooperative end-to-end self-management. Coupling is beneficial particularly in cases of congestion at the in-transit level, which normally occur at one of the shared links in the data path between the sources and sink nodes. In the standalone case (without cooperative management), i.e. if application level management was used in isolation, the application level controller would detect the congestion and advise the service manager to reduce the amount of data sent on the network and increase the amount of data written to the local storage thereby avoiding data loss. While this would eventually reduce the congestion in the data path, it would require that the data blocks written to the local storage be manually transferred to and processed at the sink. However in the coupled scenario the in-transit node signals the controller at the application in response to local congestion that it detects by observing its buffer occupancy. This would in-turn allow the application level controller to detect congestion more rapidly, rather than waiting until the congestion propagates back to the source and in response, it increases the slack on the data item to a higher value. The would enable future in-transit functions on application level data items to be provisioned to take care of network congestion. The controller also throttle items in its buffer till the congestion at the in-transit nodes is relieved. This, in turn, reduces the amount of data that is written to the local disk at the source and improves quality of data items reaching the sink.

1.5 Contributions and Impact of the Research

The research presented has the potential to significantly impact how Grid application workflows are formulated and managed. The broader impacts flow from a novel development paradigm for resolving key complexities of distributed systems by (1) identifying key challenges of the emerging self-managing software approaches and (2) developing novel solutions using a combination of heuristics and systematically sound theoretical techniques to design and deploy large-scale self-managing computer systems. Using a theoretic basis for self-management will provide a robust foundation, in contrast to current largely ad hoc and heuristics-based approaches, allowing developers to reason about performance and reliability behaviors and guarantees. This will, in turn, impact a range of distributed applications including science/engineering applications, business applications, and more generally, applications involving sensing, collection, analysis and dissemination of information.

1.5.1 Contributions

Self-Managing Distributed Application Workflows:
The primary contribution of this research is the development of a cooperative two self-managing data streaming and in-transit processing framework for application workflows. The first level of this framework uses proactive application level management for data streaming while the second level uses reactive and opportunistic strategies for data streaming and processing respectively. Proactive and Reactive strategies at both layers work in tandem to satisfy end-to-end QoS requirements of the application.

The specific contributions of this thesis include:

- **Design of the Two level self-managing framework:** Two level self-managing framework which addresses end to end and in-network QoS management, forms the basis for adaptive scientific workflows for processing data in-transit.

- **Design of the slack metric:** The slack metric generated at the application level is used for provisioning in-transit processing. The slack metric captures QoS of the application data from end-to-end and guides the processing and forwarding of the data at the shared in-transit nodes.

- **Adaptive buffer management for data streaming:** Adaptive buffer management scheme transfers data from live simulations or from in-transit functions running in batch or interactively either on a remote supercomputer or shared in-transit nodes over a WAN as efficiently as possible and provides minimal overhead on the simulation or in-transit nodes.

- **"Self-Managing" data streaming using policy based mechanisms:** Policy based programming framework introduces adaptive behaviours into the data streaming service and takes into account the key characteristics of unreliable execution environments.

- **"Self-Managing" data streaming using policy and model based online control:** A combination of rule-based self-management approaches with formal model-based online control strategies produces adaptive behaviour in data streaming applications.

1.5.2 Impact of the Presented Approach

The two level cooperative framework can be used in various scientific and financial application workflows to achieve self-managing behaviour. A selection of potential applications of the proposed research is highlighted below.

Efficient Monitoring and Coupling of Petascale Simulations for the Scientific Discovery Process

The research approach described in this thesis enables a paradigm shift in which scientists and end-users operate on the simulation data and effectively helps them to

"find the needle in the haystack" of data, and perform complex code coupling. The framework will seamlessly enable scientists to monitor and couple codes, and to move large amount of data from one location to another with low overhead on executing simulations. Further it will empower scientists to ask "what-if" questions and in-turn provide answers to these questions in a timely fashion. Furthermore these techniques will enable effective data management which will not only just become important-it will become absolutely essential as current simulation and execution environments move beyond current petascale system into the age of exascale computing. Furthermore as simulations increase in size and complexity, the main computational part of the simulation will need to be scaled to larger machines, while the I/O routines need to be transported to specialized machines such as GPGPUs or cheaper machines like clusters and processed with the help of the framework discussed in this thesis.

Mathematical Programming Techniques

This thesis will allow programmers and application scientists to use mathematical programming techniques and frameworks to achieve application-level adaptation in distributed environments. Furthermore these techniques can be applied to a wide range of simulations including financial applications where the data production rates are stable due to their inherent load balanced and parallel nature. These novel solutions based on systematic and theoretically sound techniques can be used to design and deploy large-scale self-managing distributed application workflows whose correctness can be analyzed prior to deployment. Using a control-theoretic basis for self-management will provide a robust foundation, in contrast to current largely ad-hoc and heuristics based approaches, allowing developers to reason about performance and reliability behaviors and guarantees. This will, in turn, impact a range of distributed applications including science/engineering applications, business applications, and more generally, applications involving sensing, collection, analysis, distribution of information, and monitoring to control environments.

1.6 Outline of the Thesis

The rest of this thesis is organized into various chapters as follows.

Chapter 2 outlines the related work of the thesis.

Chapter 3 outlines the architecture of the two level self-managing framework. It also presents the problem formulation for data streaming and in-transit processing in scientific workflows which forms the basis for the design of the two level self managing framework. The next three chapters describe various techniques for self managing data streaming at the application level. The final two chapters before the conclusion describe various techniques for management of in-transit processing. They also present details of the cooperative management framework for self-managing application workflows.

Chapter 4 describes our initial approach of data streaming through the use of adaptive buffer management for scientific simulations. This technique for buffer management is used at the in-transit level for data forwarding due its low overhead on executing in-transit functions.

Chapter 5 presents a self-managing data streaming service at the application level designed using heuristics or a rule based programming system.

Chapter 6 enhances the self-managing data streaming which were rule based and tightly coupled to applications, to include model based online control.

Chapter 7 presents in-transit level data processing using reactive strategies on static data paths between source and destination to aid maximum in-transit processing on the flow. These reactive strategies were coupled application level self-managing data streaming discussed in Chapter 6 at end points to achieve good QoS on Grid workflows.

Chapter 8 presents a novel technique for provisioning in-transit processing using a slack metric generated at the application level. The slack metric approximates end-to-end constraints and guarantees "in-time" delivery of data. Furthermore it minimizes

storage and processing requirements at end points.

Chapter 9 present conclusions and outlines the future work of this thesis.

Chapter 2
Background and Related Work

Related work in self managing data streaming for scientific workflows has been divided into two sections. The first section describes the on going work in data streaming and QoS management for workflows, the second section describes work in model and mechanisms for self management.

2.1 Data Streaming in Scientific Workflows

Data Grids [27] and related research efforts such as SRB [74, 89] and SRM [79] have focused on the management and transport of large volumes of data for visualization and analysis. Traditionally data movement on Grids have been done using specialized protocols such as GridFTP [8] and the Globus XIO API [7], which define file manipulation and file transfer protocols for general-purpose secure, reliable data movement in Data Grids. These efforts have focused on file-based data movement and data post-processing rather than data streaming and in-transit data manipulations. There is existing work for scientific data manipulation such as DataCutter [14] which focuses on the constraints of partitioning for wide area and out-of-core computation. Also, efforts such as [43, 78] have investigated other techniques of streaming data from scientific simulations for analysis and visualization. The systems that are most related to this work are adaptive Grid workflow systems, such as Active Buffering [62] and Autoflow [76], address issues of data streaming and/or in-transit processing. Our approach however differs from these efforts in that it develops a cooperative two level integrated approach that is specifically targeted to address both data streaming and

in-transit data processing challenges for Grid workflows.

2.1.1 High Throughput Data Movement using Specialized Protocols

This section describes related work in data movement using specialized protocols in detail. Many of these protocols could be used with the adaptive buffer management (as discussed in Chapter 4) techniques to yield high data throughput in scientific simulations. Some Grid based protocols which are directly related to our work include:

SABUL: SABUL [38] is an application-level data transfer protocol for data-intensive applications over high bandwidth-delay product networks such as ESnet [50]. SABUL was designed for reliability, high performance, fairness and stability. It uses UDP to transfer simulation data and TCP to return control messages. A rate-based congestion control that tunes the interpacket transmission time helps achieve both efficiency and fairness. In order to remove the fairness bias between flows with different network delays, SABUL adjusts its sending rate at uniform intervals, instead of at intervals determined by round trip time. SABUL has demonstrated its efficiency and fairness in both experimental and practical applications. SABUL has been implemented as an open source C++ library, which has been successfully used in several Grid computing applications.

bbcp: bbcp [39] is a point-to-point network file copy application written by Andy Hanushevsky at SLAC as a tool for the BaBar collaboration. It is capable of transferring files at approaching line speeds in the WAN. BBCP is currently in alpha at the time of this writing and is useful for transferring files through high bandwidth links. bbcp is available on machines such as Jacquard [65] supercomputer at NERSC as an alternative to secure copy ("scp").

GridFTP: GridFTP is a protocol defined by Global Grid Forum Recommendation (GFD.020), RFC 959, RFC 2228, RFC 2389, and a draft before the IETF FTP

working group. The GridFTP protocol provides for secure, robust, fast and efficient transfer of (especially bulk) data. GridFTP provides reliable file transfer service, and can be used to build Replica Location Service (RLS). Another important feature of GridFTP is parallel data transfer on wide-area links, through the use of multiple TCP streams in parallel (even between the same source and destination). It is found to improve the aggregate bandwidth between endpoints over using a single TCP stream. The other key features of GridFTP include striped data transfer, partial file transfer, support for reliable and restartable data transfers and Grid Security Infrastructure (GSI) and Kerberos support for data security. GridFTP could derive benefits of our work on model-based online control (refer to Chapter 6) to predict application and environment behaviour to enable self-managing data streaming.

IBP: The Internet Backplane Protocol (IBP) [71] is a middleware for managing and using remote storage. The design of IBP is shaped by analogy with the design of IP in order to produce a common storage service with similar characteristics. Though it has been implemented as an overlay on TCP/IP, it represents the foundational layer of the "network storage stack". Just as IP datagram service is a more abstract service based on link-layer packet delivery, so is IBP, a more abstract service based on blocks of data (on disk, memory, tape or other media) that are managed as "byte arrays" By masking the details of the local disk storage fixed block size, different failure modes, local addressing schemes this byte array abstraction allows a uniform IBP model to be applied to storage resources generally. The use of IP networking to access IBP storage resources creates a globally accessible storage service. IBP was used in the work on adaptive buffering and was found be achieve low streaming overhead on scientific simulations with high data generating rates.

Unfortunately none of these systems address issues directly related with self-managing aspects of data streaming for scientific workflows.

2.1.2 QoS Management in Data Intensive Workflows

Recent research efforts on data-intensive scientific workflows include BioOpera [20] Pegasus [30], Sphinx [42], GridBus [85] GridAnt [51] and myGrid [82]. All these efforts focus on constructing end-to-end applications on the Grid. These efforts have addressed QoS management issues in workflows using adaptive reservation and preallocation of resources, cost based scheduling, brokering, negotiation (Grid Quality of Service Management(G-QoSm)) and using publish subscribe for notification services. Similarly workflow systems like Kepler [60], Discovery NET [75] and Triana [83], provide mature and generic platform for building and executing workflows, and support multiple models of computation. These systems however leave QoS management issues to the implementer of the workflow and are complementary to this work. QoS management has also been addressed by general multimedia and business workflows using game theoretic framework for incentives [22], microeconomic flow control techniques [36], and multi-agent scheduling mechanisms [21] where adaptive pricing is used. Medusa [10] a distributed stream processing system uses private pair wise contracts for managing QoS issues. While these efforts are related, their target workflows differ significantly from the data intensive workflows of high-performance scientific applications in their data size.

2.2 Model and Mechanisms for Self Management

This section presents related work involving self-managing applications, which is divided into rule based adaptation and control based adaptation.

2.2.1 Rule-based Adaptation of Application Behavior

Application or service adaptation using rule-based techniques was systematically studied in Accord and applied to objects [53], components [55], [68] and Grid services [54] for scientific applications and workflows. Active buffering, a buffering

scheme for collective I/O, in which processors actively organize their idle memory into a hierarchy of buffers for periodic data output using heuristics was studied in [62], [63]. RESAS [19] was one of the early systems to support dynamic rule-based adaptation of real-time software and provides tools for programmers. Specifically, it provides algorithms to modify the reliability and/or timeliness of software without affecting other aspects of its functionality. A key challenge in rule-based adaptations is the generation of rules, which is typically manual. Correctness of rule-based management has been investigated for business applications using complex mechanisms based on databases [4] or business models [26], and in the security domain using types [61] as part of the policy specification process and using auctions [23] at runtime.

2.2.2 Control-based Adaptation of Application Behavior

Recent research efforts [40], [41] have investigated using feedback (or reactive) control for resource and performance management for single-processor computing applications. These techniques observe the current application state and take corrective action to achieve specified QoS, and have been successfully applied to problems such as task scheduling [24], [58] bandwidth allocation and QoS adaptation in web servers [3], load balancing in e-mail and file servers [40], [57], [70] network flow control [64], [81] and processor power management [59], [77]. Feedback control theory was similarly applied to data streams and log processing for controlling the queue length and for load balancing [90]. Classical feedback control, however, has some inherent limitations. It usually assumes a linear and discrete-time model for system dynamics with an unconstrained state space, and a continuous input and output domain. The objective of the research presented in this thesis is to address this limitation and manage the performance of distributed applications that exhibit hybrid behaviors comprised of both discrete-event and time-based dynamics [2], and execute under explicit operating constraints using the proposed LLC method. Predictive and change-point detection algorithms have been proposed for managing application performance, primarily to

estimate key performance parameters such as achieved response time, throughput, etc., and predict corresponding threshold violations [86]. The approach used in this thesis combines rule based adaptations with mathematically sound models and optimization techniques to achieve self-management in distributed applications.

Chapter 3

Two Level Self-Managing Framework

This chapter formally poses the problem of data streaming and in-transit processing for scientific workflows and then illustrates the two level self-managing framework for addressing this problem.

3.1 Problem Formulation for Data Streaming and In-Transit Processing

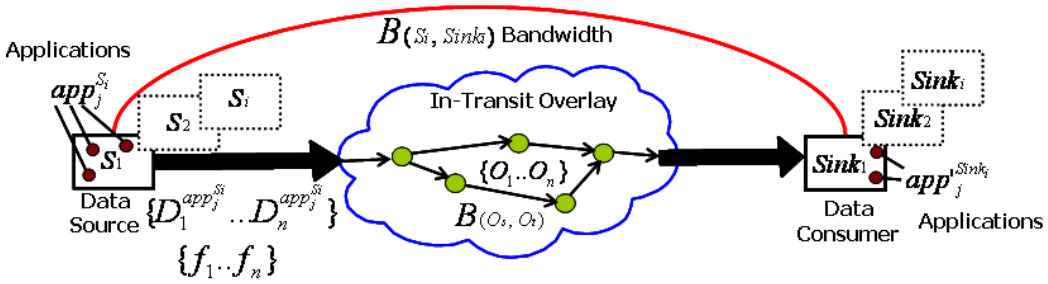

Figure 3.1: Problem Formulation for Data Streaming and In-Transit Processing

Consider a set of data generating applications $app_i..app_n$ executing on resources $S_1..S_n$ termed as data sources in our workflow. Each of these applications belong to a particular source S_i and are denoted as $app_j^{S_i}$. Similarly applications that execute on resources called sinks $Sink_1..Sink_n$ are denoted as $appl'^{Sink_i}_j$ and have one to one correspondence with applications executing at the source. $app_j^{S_i}$ generates data $D_1..D_n$ and at various time intervals $1..n$ depending on the nature of the application. Each of these data items $D_k^{app_j^{S_i}}$ need to be processed with functions $f_1..f_n$ which will be applied on them either at the in-transit overlay resources $O_1..O_n$ or at the

sink/destination if the in-transit fail to operate on the data items. An important assumption in our system is that $size(D_k^{app_j^{S_i}})$ is unchanged when functions $f_1..f_n$ are applied on them. These in-transit functions can be identical or varied ($f_1 \neq f_2 = \ldots f_n$). The in-transit functions f_r are also applied on the whole data items $D_k^{app_j^{S_i}}$. The number of in-transit functions applied on the data item is termed as the quality of the data, $Qual_{D_k^{app_j^{S_i}}}$. The bandwidth for end-to-end data streaming for applications executing on sources and sinks through the in-transit overlay is denoted as $B_{S_i,Sink_i}$. The goal of the data streaming and in-transit manipulation framework is to allow maximum $max(D_k^{app_j^{S_i}})$ data items to reach the sink, within a strict end-to-end time window of $Tendtoend_k^{D^{app_j^{S_i}}}$ depending on the application characteristics at the data source and sink, with maximum $max(Qual_{D_k^{app_j^{S_i}}})$ using resources in the in-transit overlay. The basic goal is for the processed data to arrive just "in-time", as faster arrival of processed/unprocessed data leads to storage problems at the sink and slower arrival of processed/unprocessed data leads to QoS issues for applications $appl_j'^{Sink_i}$ executing on the sink. It is assumed here that the latency for forwarding data items between overlay nodes is smaller compared to end-end latency in other words $B_{O_s,O_t} \leq B_{S_i,Sink_i}$. In other words, end-to-end QoS objectives at the application and sink are to ensure:

Maximizing:

$$Qual_{D_k^{app_j^{S_i}}} \& size(D_k^{app_j^{S_i}})$$

Subject to:

$$0 \leq Ttransit_k^{D^{app_j^{S_i}}} \leq Tendtoend_k^{D^{app_j^{S_i}}} \forall i,j,k$$

where $Ttransit_k^{D^{app_j^{S_i}}}$ is composed of $Tproc$, $Tbuff$ and $Tforward$ at the overlay O_s. Therefore

$$0 \leq \sum_{r=1}^{n}\sum_{s=1}^{n} Tproc_{k,f_r,O_s}^{D^{app_j^{S_i}}} + \sum_{s=1}^{n}(Tforward_{k,O_s}^{D^{app_j^{S_i}}} + Tbuff_{k,O_s}^{D^{app_j^{S_i}}}) \leq Tendtoend_k^{D^{app_j^{S_i}}} \forall i,j,k$$

Additionally the slack metric denoted as $slack(D_k^{app_j^{S_i}})$ is used to capture provisioning of in-transit processing on the data items and is initially approximated using time for end-to-end forwarding and processing entirely at the sink $Sink_i$ (obtained through history). In other words:

$$slack_{initial}(D_k^{app_j^{S_i}}) = size(D_k^{app_j^{S_i}})/B_{S_i,Sink_i} + \sum_{r=1}^{n} Tproc_{fr,Sink_i}^{D^{app_j^{S_i}}} \forall i,j,k$$

Each operation on the data item in the overlay which includes either processing, buffering and forwarding updates the slack metric. A negative slack metric value indicates that data items $D_k^{app_j^{S_i}}$ reached the sink later than expected (or $slack_{initial}(D_k^{app_j^{S_i}})$ had estimated), similarly a positive value indicates the earlier arrival of data at the sink. Slack metric values are later corrected through feedback at the sink to include time to buffer data items in the overlay nodes O_s. The goal at the in-transit nodes is that the slack metric on data items should have near zero value when it reaches the sink ($slack(D_k^{app_j^{S_i}}) \approx 0$).

3.2 Research Approach: Two Level Self-Managing Framework for Data Streaming and In-Transit Processing

To address the problem of self-managing data streaming and in-transit processing discussed in the previous section a research approach consisting of two a level self-managing framework (as illustrated in Figure 3.2) is designed to address QoS issues in data intensive scientific workflows operating on the Grid. It consists of two key components: The first is an application level self-managing data streaming service, which provides adaptive buffer management mechanisms and proactive QoS management strategies based on online control and user-defined polices, at application end-points. The application level component also captures constraints for in-transit

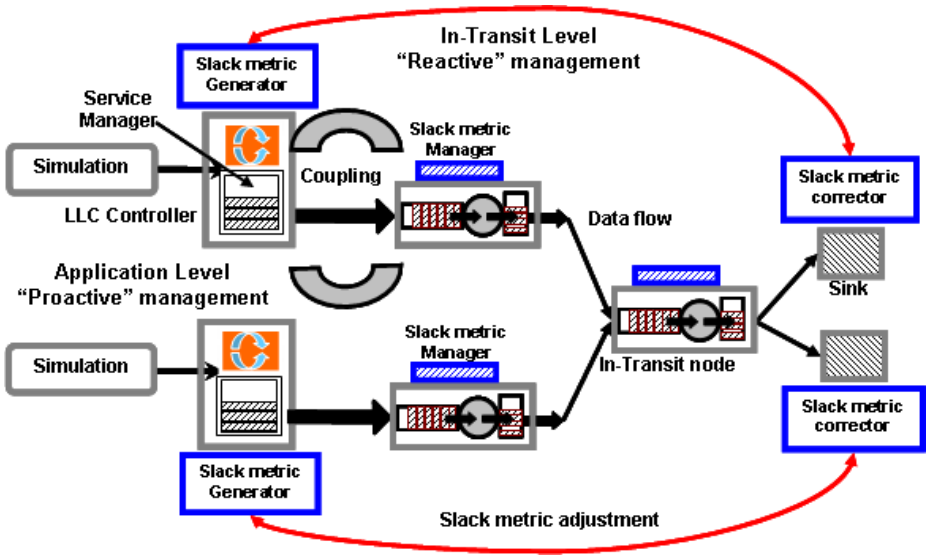

Figure 3.2: Two Level Self-Managing Cooperative Framework

processing using a "slack metric" generated at the application level. The second component provides scheduling mechanisms and adaptive runtime management strategies for in-transit data manipulation and transformation. These two components work cooperatively to address the overall application constraints and QoS requirements.

3.2.1 Application Level Self-Managing Data Streaming

The application level self-managing data streaming service combines model-based limited look-ahead controllers (LLC) and rule-based autonomic managers with adaptive multi-threaded buffer management and data transport mechanisms at the application endpoints. It is constructed using the Accord-WS infrastructure (discussed in detail in Chapter 5) for self-managing Grid services and supports high throughput, low latency, robust application level data streaming in wide-area Grid environments. The autonomic data streaming service consists of a service manager coupled with an LLC controller. The service manager monitors the state of the service and its execution context, collects and reports runtime information, and enforces the adaptation actions determined by the controller. Augmenting the element manager with an LLC controller allows human defined adaptation polices, which may be error-prone

and incomplete, with mathematically sound models and optimization techniques for more robust self-management. Specifically, the controller decides when and how to adapt the application behavior and the service managers focus on enforcing these adaptations in a consistent and efficient manner. Additionally the LLC controller is enhanced with a slack metric generator which fixes a deadline for delivery of simulation data at the sink. The input from the LLC controller, slack correctors at the sink and in-transit slack metric managers or SLAM's, drive the slack generator at the application level. Chapter 4 , Chapter 6 and Chapter 5 discuss about the self-managing Application Level data streaming in detail.

3.2.2 In-Transit Level Data Processing

The in-transit data manipulation framework consists of a dynamic overlay of available in-transit processing nodes (e.g., workstations or small to medium clusters) with heterogeneous capabilities and loads. Note that these nodes may be shared across multiple workflows. Each node can perform a limited number of operations on the in-transit data, these include processing, buffering and forwarding. The slack metric managers at the in-transit level update the slack metric generated at the application level after each of the following operations on the data. The processing on the in-transit data depends on the capacity and capability of the node, the amount of processing that is still required on the data which is depicted as the slack metric (generated at the application level using the slack metric generator), the network conditions between the in-transit nodes and the load on a particular in-transit node. The amount of processing completed is logged in the data block itself. The goal of the in-transit processing is to process as much data as possible before the data reaches the sink. A processing that is not completed in-transit will have to be performed at the sink. The current design of the framework assumes that each node can perform any of the required data manipulations functions. It also assumes that the in-transit functions do not change the size of the data items during in-transit. Chapter 7 and

Chapter 8 discuss in detail about the In-Transit level data processing using reactive and slack metric based mechanisms.

Chapter 4

Data Streaming using Adaptive Buffer Management

Large scale simulations are increasingly important in many fields of science. The research described in this chapter grew from the requirement to deal with the output of a major fusion plasma simulation, the Gyrokinetic Toroidal Code (GTC) [52]. This code examines the highly complex, non linear dynamics of plasma turbulence using direct numerical simulations, and currently generates about 1TB/week of simulation results data during production use.

We have developed a system which efficiently and automatically transfers chunks of data from the simulation to a local analysis cluster during execution. By overlapping the simulation with the data transfer and with the analysis, scientists can analyze their results as they are being produced.

The rate at which fusion scientists generate data from their simulations today is about 1 TB/week, but we expect this figure to increase by an order of magnitude in the next five years. The conventional trend has been to place the generated computational data on the supercomputing sites and later transfer the data manually, or, to execute remote visualization and post-processing of the data. Both approaches encounter difficulty, forcing scientists to concentrate on data transfer and remote visualization issues rather than dealing with the physics. Remote visualization in particular raises issues of latency and network quality of service. To overcome these challenges we develop a low overhead threaded parallel buffer to transfer data from simulations to the scientist's local computing cluster(s) where access to the data is most convenient

and efficient.

The driving force of the threaded buffer for data transfer has been to provide a minimal overhead in simulations while utilizing network resources to the maximum. The application uses simple APIs to activate the transfer. To make this data transfer efficient with the added advantage of global scheduling, optimization of data movement, storage and computation we exploit Logistical Networking (LN) [11] built on the Internet Backplane Protocol (IBP). LN allows for a flexible sharing and utilization of writable storage as a network resource, which is our natural choice for data flow in a data "pipeline" [49] with various depots (storage) locations containing the data in various stages of transformation. The existence of pervasive depots aids in the creation of a reliable data pipelines. It allows simulations to transparently store data to adjacent depots in case of network failures at the receiving end or buffer overflows at the sending end. Post-processing applications can automatically pull/fetch this data through an alternate path from depots adjacent to the computing sites, as the data is transferred from the simulations. This two-way push and pull mechanism enables us to utilize the network bandwidth maximally and affect the simulation's performance minimally.

In this chapter we discuss our method of real time data streaming of the simulation data through our threaded buffer, buffer management algorithm, and transformations of the data. Our system creates a high performance data pipeline [9, 80] which enables a more efficient interaction of the scientist with the data. We discuss the various fault tolerant mechanisms used in case of buffer overflow or network failures.

The chapter is divided into the following sections. Section 4.1 discusses the GTC workflow and the data pipeline for scientific simulations. Section 4.2 discusses the threaded buffering scheme; Section 4.3 elaborates on the implementation, operation and fault tolerance mechanism of the threaded buffer scheme using the Logistical Networking (LN) technology. Section 4.4 discusses the experimental evaluation of the adaptive buffering scheme. Section 4.5 discusses future work and conclusions.

4.1 Automation of the GTC Data Pipeline

The need for computer aided tools increases with size and complexity of the simulation generating the data. Without automation, Scientists spend a large portion of their time managing the workflow and data flow. Such management includes organizing and sharing raw and derived data between collaborators, transforming data formats, etc. In this section we would like to illustrate a general data flow pattern of our GTC simulation which runs in parallel on a supercomputer at NERSC, and how this data undergoes continuous transformation until it reaches the desktops of scientist's collaborating with PPPL in analyzing the simulation data. We consider PPPL as one transformation point as it flows along to other collaborators. Figure 4.1 illustrates

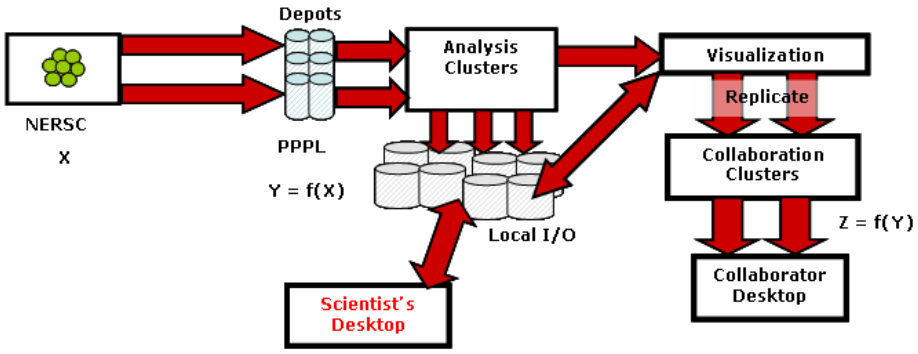

Figure 4.1: Data Pipeline for the GTC simulation

the end to end data pipeline used by the GTC simulation running at NERSC. The simulated data is transferred concurrently as the simulation is taking place through our buffering mechanism. The raw data (X) streams over to a data analysis center and it is converted into appropriate formats (e.g. HDF5 or NetCDF) as required by the scientists (scientists can specify the format in which they want to transform the data using simple APIs in their codes). The analysis clusters start converting the raw simulation data to an appropriate format for visualization as soon as the first time-step arrives. The converted data (Y) is written to disk and fed into visualization routines. This data flow scheme is particularly well suited for the analysis of fusion codes as this makes efficient use of dedicated computing resources at the scientists'

local resources and additionally provides the scientists with real-time visualization capability for their simulations. Finally at the end of this data flow, the data reaches the desktops of the collaborators working on the fusion codes who may then further transform the data (Z).

4.2 Design of the Threaded Buffer Data Streaming

The goal of the buffering scheme is to transfer data from a live simulation running in batch on a remote supercomputer over a Wide Area Network (WAN) to our local analysis/visualization cluster as efficiently as possible and provide minimal overhead on the simulation [31, 32, 63]. It should also have replication abilities so that the processed data can be duplicated to collaborators' clusters as and when needed. To avoid loss of raw data either due to buffer overflows (when the generated data does not fit into the buffer) or network failures, the data should be transferred fault tolerantly.

To achieve this data transfer we use a buffering algorithm that uses a circular queue and a threaded queue manager (one for each node of the supercomputer) so that it performs wide-area data transfer with minimal memory overhead on our simulations. This buffering mechanism copies the simulation data to a small memory buffer which is allocated by the user in his/her simulation. The buffer can be thought of as a queue of data blocks expecting to be transferred. This queue is circular, thus it wraps around after it reaches the end. Each data block generated by the user can have varying sizes but the queue manager chops the data into a uniform block size, which is configurable by the user. The queue manager maintains two pointers within the buffer. The first is the *write position*, which is the position where the data is being copied into the buffer (i.e. where the simulation writes data into the buffer). The second is the *send position* to indicate the current position in the buffer where the transfer mechanism is operating (position of last successful transfer). The *send position* changes in multiples of blocksizes. The user can append small pieces of

information to the data that contains information for the post-processing routine to operate on data (i.e. metadata). In practice the metadata added to the data never exceeds a small number of bytes and forms a tiny fraction of the actual data to be transferred. The queue manager adds metadata to the data before placing the data on the buffer. The queue manager then updates the values of the *send position* and *write position* whenever data is transferred out of or added to the buffer. After the data is transferred and the *send position* is moved, the application can write into that space. In the next section, we describe the simple buffer management scheme which adapts to the network conditions.

4.2.1 Adaptive Buffer Management

We use a simple algorithm to manage the buffer that adapts to both the computation's output rate and network conditions. First, we recognize that the simulation is based on a series of time-steps. The data generation rate is the amount of data generated per step, divided by the time to perform the step. For the GTC code, this can vary from 1 to 90 Mbps, depending on simulation and analysis options.

We also recognize that the network connectivity between the supercomputer and the analysis cluster places an upper limit on the transfer throughput. The smallest pipe between the supercomputer and the analysis cluster will determine the theoretical maximum throughput for the transfer. Since the transfer routines use TCP for reliable data transfers, we understand that we will get even less than the theoretical throughput [50]. The algorithm tries to dynamically adjust to the data generation rate and the available network rate. It does this by sending all the data that has accumulated since the start of the last data transfer. If the data generation rate exceeds the transfer rate, more data will be in the buffer. In this case, the queue manager will increase the amount of multi-threading in the transfer routines to improve throughput. If the transfer rate exceeds the data generation rate, then less data will appear in the buffer for the next transfer. The queue manager will then reduce consumption

of unnecessary network resources. The initial transfer begins after the first time-step is output. All subsequent transfers start as soon as the prior transfer ends.

After some number of time-steps, if the network is stable and the data generation rate is less than the network transfer capacity, then the queue manager tends to reach equilibrium and match the transfer rate to the data generation rate.

Several buffer management states occur, depending on the relationship between data generation and data transfer rates, as is described here:

- **Data generation rate exceeds transfer rate** In this state, we maximize the network throughput and move as much of the data to the analysis cluster as possible. In the adaptive buffer transfer mechanism we use the input from the previous step (state) while sending data in the next step and form a loose feedback mechanism. We send the excess data that cannot be transferred to nearby disk and signal the receiving process of this data to start re-fetching this data using any remaining bandwidth, or after completion of the simulation. The queue manager detects this if the simulation needs to write data to the buffer, but the write position is too close to the send position which indicates that there is not enough space in the buffer for the new output.

  This makes our scheme "network aware" as our transfers are dependent on the network on which we are operating and the blocks sent out during each transfer depend of the previous transfer.

- **Data generation and transfer rates are similar** In this situation, significant new data accumulates in the buffer during each transfer. The size of the first transfer is one block. Subsequent transfers usually involve a larger number of blocks. These multi-block transfers use multiple IBP threads and can consume available network capacity.

- **Data generation rate is small compared to transfer rate** If data is generated at a rate in which after every transfer the scheme finds the buffer empty

it waits and does nothing till more data is generated in the buffer. In this state the buffering scheme would send out block by block, using minimal network resources.

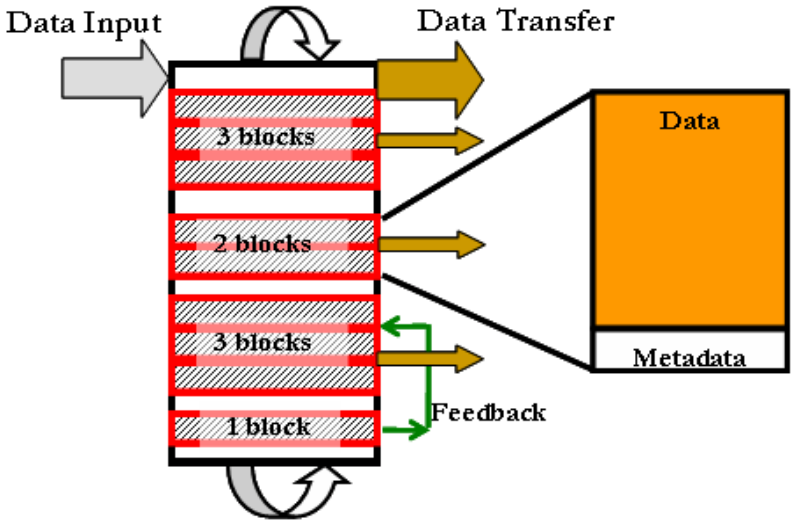

Figure 4.2: Adaptive Buffer Management Scheme

Figure 4.2 shows the adaptive buffer management scheme that we use in this chapter. This "latency aware" transfer mechanism is particularly useful in cases where blocks are generated quickly around 65-75Mbps as compared to the simple buffer scheme which sends each individual piece in the buffer. It is powerful in cases where data is generated slowly (i.e. less than 1Mbps), in this case if the block size is set to 1MB we send just a single block of data continuously. We believe that that this feedback-based buffer management scheme improves the transfer mechanism by sending as much data as the network can handle and caching the rest to disk until the end of the simulation run. It takes decisions based on the previous transfer when deciding which blocks to transfer and which blocks to write locally. The scheme illustrated in Figure 4.2 works well for transferring date from the simulations at NERSC to PPPL and easily saturates the link as will be shown in the Section 4.4.

4.2.2 Usage of Buffering Scheme

To take advantage of our transfer mechanism, the application first makes calls to `t_open()`, which initializes a finite buffer and the queue manager. The queue manager will then wait for any data generated by the simulation. The user then inserts `t_write()` statements at appropriate places in his code where data is generated. The t_`write()` statements copy the generated data to the buffer initialized the user. To close and flush the buffer at the end of the simulation, the application uses `t_close()`. The application can also specify certain information about the data which will be useful for post-processing, by using a `write_metadata()` statement in conjunction with the `t_write()` statements. This statement is useful for starting post-processing at the raw data receiving end. Metadata for the data transfer include global and local dimensions for the global array which will be required for "HDF5" or "NETCDF" or "ASCII" file creation, name of the variables transferred in the data block, name of the final generated file. Metadata size is typically in the order of few hundred bytes.

4.3 Implementation of the Adaptive Buffering Scheme

4.3.1 Building Block

In this section we present the design and implementation of the adaptive buffering scheme using LN which forms the basic building block. Logistical Networking (LN) [11] refers to the global scheduling and optimization of data movement, storage, and computation based on a model that takes into account all of the network's physical resources.

Logistical Networking (LN)

Unlike traditional networking, which does not explicitly model storage or computational resources in the network, LN offers a general way of using computing resources

to create a common distributed storage infrastructure that can share out storage and computation the way the current network shares out bandwidth. The middleware components that enable logistical networking are arranged in the "network storage stack," [11] analogous to the IP stack, using a bottom-up and layered design approach that provides maximum scalability. Components of the network storage stack are described below bottom up:

IBP - Internet Backplane Protocol: IBP is the foundation of the network storage stack and provides a highly scalable, low-level mechanism for managing network storage resources, through shared use of lightweight, time-limited allocations on storage "depots".

exNode - External Node: Similar to the concept of an inode in UNIX file systems, this is a generalized data structure which holds the metadata necessary to manage distributed content stored on IBP depots and allow file-like structuring of stored data.

L-Bone - Logistical Backbone: Directory and resource discovery service cataloguing registered IBP storage depots world-wide.

LoRS - Logistical Runtime System: The LoRS software suite integrates the underlying capabilities of IBP, the exNode, and the L-Bone into a streamlined tool for storing, accessing, and managing data.

Advantages of using LN

Data Replication for Fault Tolerance: The main reason for using the LN is the ability to stream buffers of data (not necessarily entire files) to multiple storage locations simultaneously for fault-tolerance. The ubiquity of IBP storage means that it is easy to stream data to a number of alternate depots close to the sender and create replicas close to remote receivers. Storing replicas in multiple locations provides fault tolerance in case of network or machine failures. Fault-tolerance through replication

is internal to the exNode. The LoRS handles retrieving from multiple replicas automatically [72].

IBP: Byte Array Abstraction: We chose IBP as the main transfer mechanism instead of a rigid transfer protocols, because IBP is a more abstract service that is interoperable with a variety of storage resources (disk, ram, etc.). IBP manages blocks of stored data as byte arrays, with details of the storage (fixed block size, differing failure modes, local addressing schemes) masked at the local level. The use of IP networking to access IBP storage resources creates a globally accessible network of storage depots.

Logistical Networking offers advantages not available elsewhere. Since Grid Protocols [6] do not support replication internally, we would have to use a higher level service such as the Replica Location Service (RLS) to track where copies of the complete files reside [28]. When retrieving the data, we would then have to determine which replica to download. If, on the other hand, we used raw sockets and wanted to implement replication for fault-tolerance, we would have to write our own servers to hold the data, write the transfer management code to use them, and design some method for tracking the replicas and reassembling the pieces-effectively recreating the LN software and infrastructure.

4.3.2 Operation of the Adaptive Buffering Scheme

The design of the streaming mechanism using our circular buffer and queue manager consists of a buffer for each processor on the simulation/computing end which generates data. The threaded write library on the sending end calls the LoRS library which ultimately transfers data using the IBP library to an IBP depot on the receiving end. After the simulation data and its metadata have been transferred, the LoRS library constructs an exNode which it returns to the queue manager. The queue manager then sends the exNode to a waiting process, *exnodercv*, in the analysis cluster at PPPL via a socket. Although this is an additional step for every transfer, the impact

is minimal and provides some benefits. First, each exNode does not exceed 10-20KB in size. Second, the exNodes (represented as XML) are transferred separately to a program on the receiving end and hence do not interfere with the main data transfer or the computation. Third, since the exNodes are represented as in an XML format they allow for platform interchangeability.

The simulations normally run in batch. The receiving part on the PPPL end consists of the *exnodercv* daemon listening for exNodes on a well known port. This program keeps track of the data transferred during the simulation and appropriately calls the post-processing routines for visualization/data transformation specified by the user. We have presently incorporated the HDF5 and ASCII routines which generate appropriate files for visualization/post-processing the simulation data. Since the post-processing routines at PPPL read the transferred data from the depots using the exNodes sent to the *exnodercv* daemon, this does not interfere with the running simulation at NERSC. Simultaneously, the post-processing routines can invoke the LoRS augmentation API which replicates the post-processed files and publishes the exNode on a well known public web server for later access by collaborators.

Failsafe Mechanisms using LN

The overall goals of our data transfer mechanism was to provide a low overhead of transfer and fault tolerance. Failures are common in the scenario of the threaded buffer transfer mechanism. The primary causes of failure include:

- **Buffer overflow at the sending end**

  This happens when the data generation rate at the simulation side far exceeds the capacity the network can sustain. This is typically the case when the data generation rate of the program exceeds the maximum network throughput, where the communication time far exceeds the computation time. Presently we are writing the data resulting from buffer overflows which cannot be transferred to our local depots in the form of binary files on NERSC General Parallel File

System (GPFS). After the files have been successfully written, a status signal for the failed transfer is sent to the *exnodercv* daemon. The status signal contains the transfer rate, size of failed transfer, and the location of the file to fetch. The daemon program then interprets the status of the failed transfers, like file size and the transfer rate to try to concurrently get the data from GPFS using GridFTP [8]. We would like to be consistent with the transfer mechanism by using LoRS for fetching the failed transfer data written to a local depot (instead of a file written to the GPFS) on the supercomputer, but presently due to security restrictions we are not able to set up a local storage depot on the supercomputer on NERSC.

- **Network connection to local depot is temporarily severed**

  The LoRS transfer mechanism might be unable to upload data to our local depots due to depot or host failures, lack of storage space, network congestion, etc. To address these issues we upload simulation data to the nearest available depots either on the supercomputer where the simulation is running or on depots located at San Diego Supercomputing Center. We then transfer/write a status/exNode generated for these types of upload to our *exnodercv* daemon/alternate depots. Since exNodes act as inodes for a network file and contain all replica information (locally and remotely stored), there is no need to separately fetch this data using any special transfer mechanism. The data is fetched from the depots only during post-processing of the data either during an HDF5, NetCDF or ASCII file creation routines.

Figure 4.3 illustrates the failsafe mechanism in case of buffer overflows at the simulating end if the data transfer rate can't keep up with the data generation rate. In this case, we write the data to GPFS. We then transfer the status/exNodes which explicitly have an error code for buffer overflow. The *exnodercv* process uses GridFTP to fetch data from GPFS at the simulation end. It is also possible that the some nodes

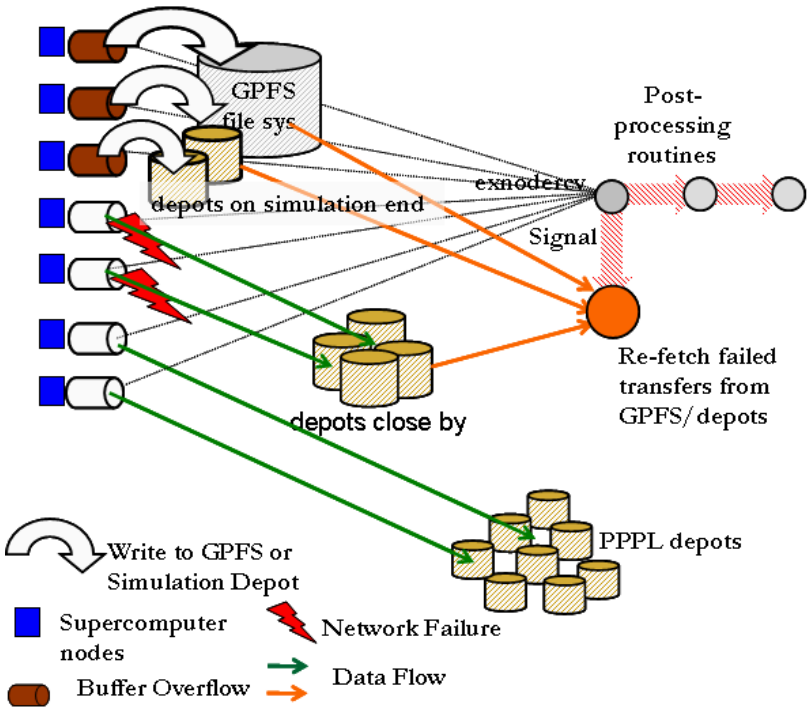

Figure 4.3: Failsafe Mechanisms using LN

in the simulation undergo a network failure/timeout. In case of a network failure or timeout of any depots at PPPL, the data is uploaded to the nearest depot using the L-Bone. In our case the nearest reliable depot to the simulation end are the depots at SDSC. We then send the exNodes/status over to our *exnodercv* process. The analysis processes read these exNodes as usual, but the read performance is less than if the data where written directly to the PPPL IBP depots.

4.4 Experimental Evaluation

The adaptive buffer management code, which we have developed, is easy to use and has simple APIs which the user can efficiently combine in his simulation to yield a high throughput data transfer. The objective of this work is that the threaded streaming should not slow down the simulation on the supercomputer (i.e. the streaming should add very little to the computation/CPU time).

To evaluate how the data transferred using this buffer and queue manager, we use

a sample program that models the GTC simulation which generates simulation data at every time step. This simulation runs on the supercomputer nodes at NERSC and the data generated is transferred to our local clusters at PPPL. We have employed buffer management with 80 MB buffers per computational node, using 1MB data block sizes. We have used a time-step as the primary reference on the X-axis (each run has 300 time-steps). The Data Generation Rates (Mbps) for each of these experiments is measured by the amount of data generated by the simulation and the time taken to generate them with no I/O involved. Data Transfer Rates is computed by the amount of MB transferred successfully divided by the time taken by the Buffering mechanism to transfer the generated data. We then study the data transfer rate (in Mbps) for various data generation rates which leads to varying data transfer sizes. Buffer overflow corresponds to data written to the local GPFS on the simulation end and must be retrieved by PPPL using the strategies described above. The block size for the transfer is 1MB. Metadata is also transferred along with the data which will be required for post-processing the simulation data.

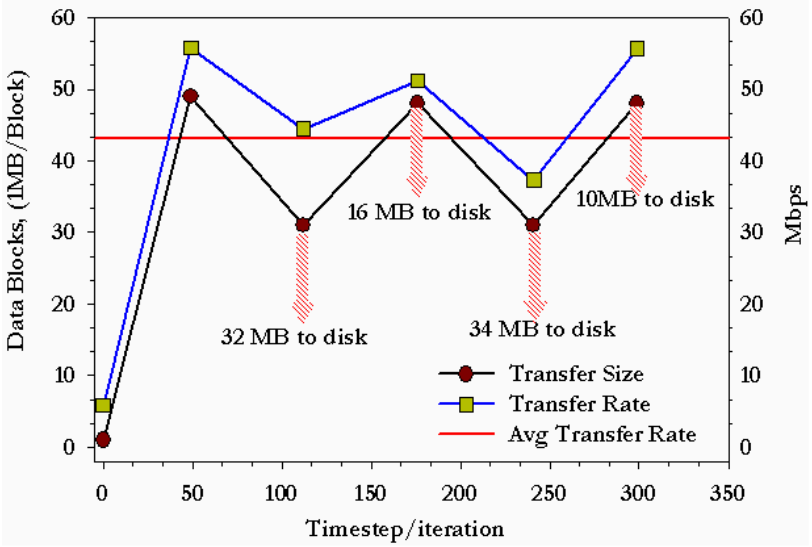

Figure 4.4: Data Streaming with 320Mbps (Buffer Overflows)

Figure 4.4 plots the blocks transferred during each timestep and the Mbps corresponding to the blocks transferred. The data generation rate for this experiment

is about 320Mbps. Our buffering scheme cannot keep up with this rate, and data is written to local disk in cases where buffer is full (80MB). The buffering scheme initially transfers the first block of data and later sends whatever is remaining in the buffer after transferring the first block. The values at data points correspond to buffer overflows since the maximum data the buffer can hold is 80MB, so when the 49 MB is being transferred data fills the buffer and (63 MB is generated out of which) 32 MB is written to disk. This process repeats itself until the simulation stops generating data. Thus the data transfer rate is around 43 Mbps. The more data that is in the buffer, the higher the chance for buffer overflow. Figure 4.5 depicts and interesting

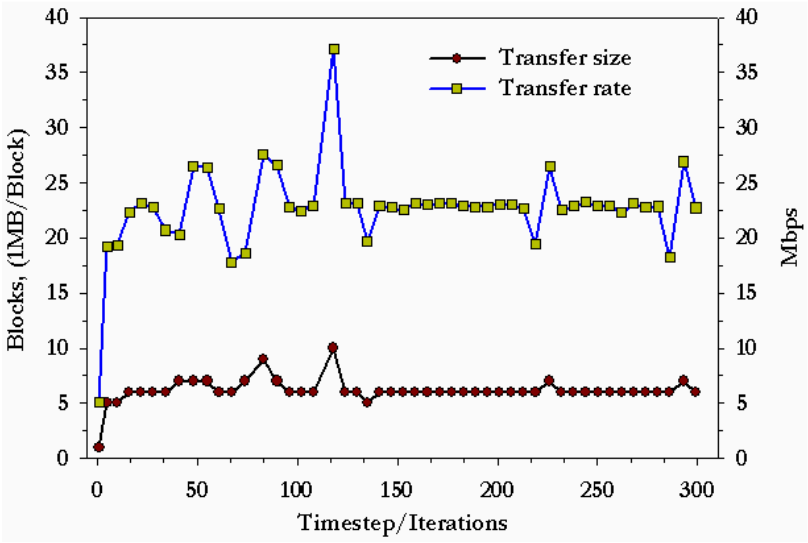

Figure 4.5: Data Streaming with 21.3 Mbps (Latency Aware)

case where data is generated at a rate of 21.3Mbps (300MB in 121 sec), all the data generated is transferred without any data written to disk or left un-transferred at the end of the simulation. The buffering scheme starts out with 1 block and then later sends out 6 data blocks but in certain cases where the rate for 6 blocks drop below 20Mbps we transfer around 8 blocks; this leads to oscillations of the data transfer block counts until around 120 timesteps when it reaches an equilibrium of 6 1MB blocks per transfer. Figure 4.6 demonstrates the network adaptability of the buffering scheme for a simulation run on two processors. Initially, the data generation rates

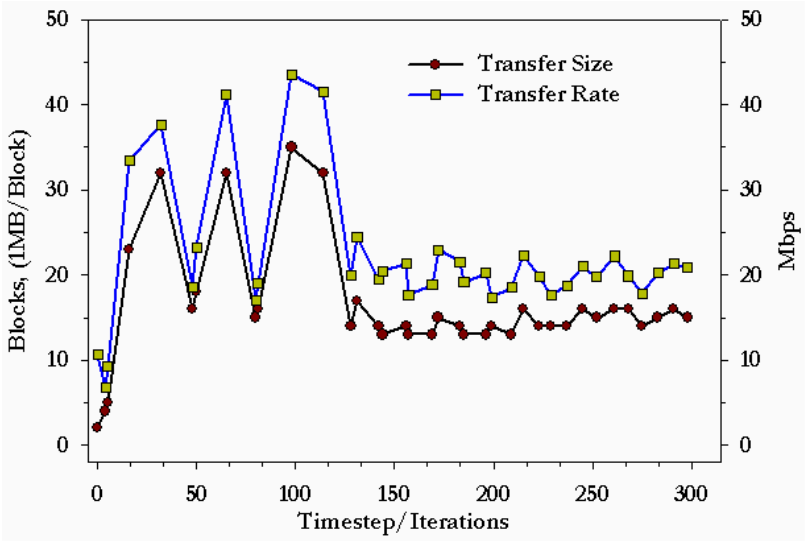

Figure 4.6: Network aware Self-Adjusting Buffer Management Scheme

(20Mbs/Processor) exceed the transfer rates. For each successive transfer, more data is available in the buffer so the queue manager sends more data and increases the level of IBP threading in the LoRS calls. The buffering scheme stabilizes itself and achieves an overall data transfer rate of approximately 20Mbps. Figure 4.7 shows the high performance buffering scheme which can keep up with rate of generation as high as 85Mbps on 32 processors. All the data generated during this period in the simulation at NERSC is transferred to our local cluster at PPPL. Figure 4.7 shows significant oscillation due to the higher number of data generator nodes involved. The best throughput that we can hope to achieve is the minimum of the data generation rate and the theoretical network throughput adjusted for TCP. The data rate is the traffic minus the headers. The maximum traffic from NERSC to PPPL is 100 Mbps, of which we hit 97 Mbps. Thus, this flow used 97% of the link (and all other users got the remaining 3%). The 100 Mbps rate assumes no one else is using the WAN connection so we can expect some value less than 100 Mbps. We can see from Figure 4.8 how the network can be easily saturated using our buffering scheme. Figure 4.8 depict the statistics when the simulation in Figure 4.7 is operational. It presents an enlarged image of the router statistics. The data rates show that we can achieve a maximum

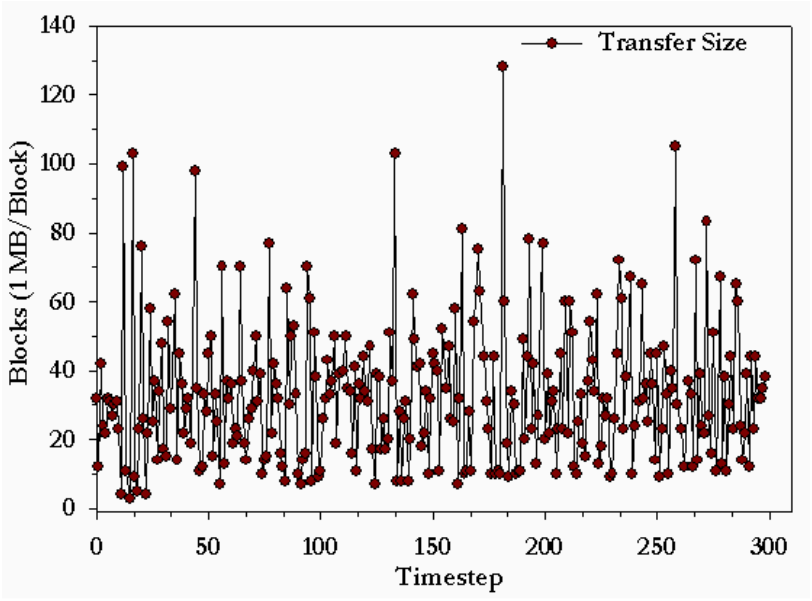

Figure 4.7: Data Generation Rate of 85mbps on 32 Nodes at NERSC Streamed to Clusters at PPPL

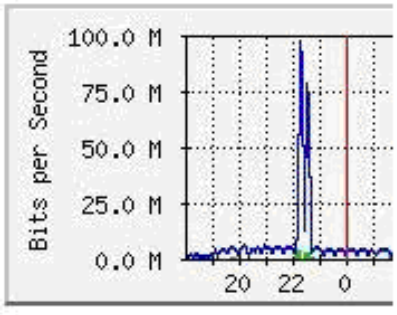

Figure 4.8: ESNET Router, Statistics (Peak Transfer Rates of 97Mbs or 100Mbs at around 22:00. Each Data Point is Calculated on a 5 Minute Average)

transfer rate of 97 Mbps as shown by the second blue spike. Figure 4.9 shows the overhead of using the buffering scheme with varying Mbps rates and compares this with writing the files to GPFS on the supercomputer nodes. We observe that in cases which are typical for present GTC codes writing data to the GPFS (2Mbps or less per node), overhead is less than for our buffering scheme 5%. In future when the GTC data generation rates are around 8Mbps, the overhead of using buffering scheme is still small. The present overhead without our buffering scheme (writing to the GPFS at NERSC [47]) is around 20% when generating hdf5 files.

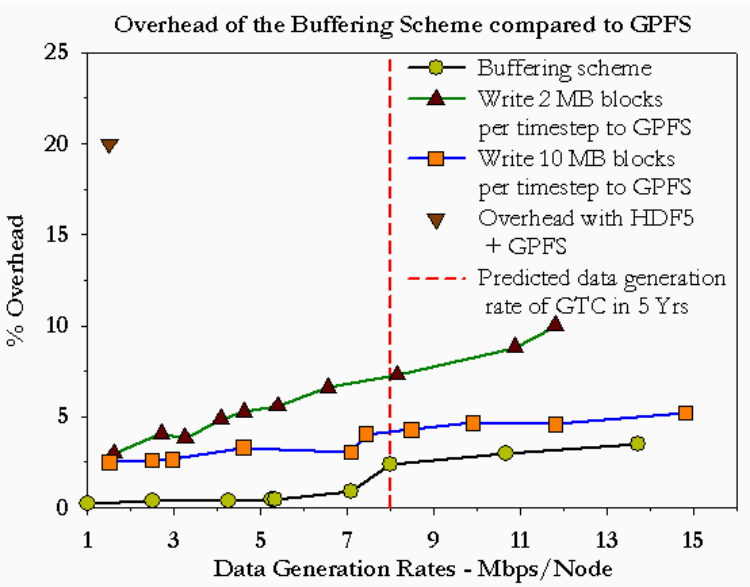

Figure 4.9: Overhead of Buffering as Compared to Writing to the General Purpose File System(GPFS) at NERSC

4.5 Conclusions

In this chapter we described development of a threaded mechanism to transfer data with a simple adaptive buffer management scheme for overlapping computation and communication. The buffering scheme had little impact on the simulation with a projected 2% overhead for codes such as GTC running on 1024 processors.

Our scheme adapts dynamically to data generation rates and network throughput, and appropriately adjusts the amount of data transferred and the level of multithreading to achieve good transfer rates. Our buffering scheme using logistical networking allows for high-performance remote transfer of data with minimal overhead on the computation system. If the data generation rate exceeds the available network resources, we have a failsafe mechanism that uses the available bandwidth to send the bulk of the data while writing the excess data locally and retrieving it later from the remote site.

In the future we will make our fault tolerance mechanism more efficient and take advantage of IBP depots within NERSC. We will work on incorporating our routines

into production runs of the GTC code. We have begun working on more optimal MxN [12] mappings for future parallel post-processing modules in our data workflow pipeline. Finally, we will incorporate priority-based transfers for optimized monitoring of selected simulation data output.

Chapter 5
Self-Managing Data Streaming using Rules

The goal of the Grid concept is to enable a new generation of applications combining intellectual and physical resources that span many disciplines and organizations, providing vastly more effective solutions to important scientific, engineering, business and government problems. The key characteristics of Grid execution environments and applications include: (1) Heterogeneity: Both Grid environments and applications aggregate multiple independent, diverse and geographically distributed elements and resources; (2) Dynamism: Grid environments are continuously changing during the lifetime of an application. Applications similarly have dynamic runtime behaviors including the organization and interactions of its elements; (3) Uncertainty: Uncertainty in Grid environment is caused by multiple factors, including dynamism that introduces unpredictable and changing behaviors, failures that have an increasing probability of occurrences as system/application scales increase, and incomplete knowledge of global state, which is intrinsic to large distributed environments; (4) Security: A key attribute of Grids is secure resource sharing across organization boundaries, which makes security a critical challenge [68].

The characteristics listed above impose requirements on the programming and management of Grid applications [69]. Grid applications must be able to detect and dynamically respond during execution to changes in both, the state of execution environment and the state and requirements of the application. This requirement suggests that (1) Grid applications should be composed from discrete, self-managing elements (components/services), which incorporate separate specifications for functional, non-functional and interaction/coordination behaviors; (2) The specifications

of computational (functional) behaviors, interaction and coordination behaviors, and non-functional behaviors (e.g. performance, fault detection and recovery, etc.) should be separated so that their combinations are compose-able; and (3) policy should be separated from mechanisms and used to orchestrate a repertoire of mechanisms to achieve context-aware adaptive runtime behaviors. Given these features, a Grid application requiring a given set of computational behaviors may be integrated with different interaction and coordination models or languages (and vice versa) and different specifications for non-functional behaviors such as fault recovery and QoS to address the dynamism and heterogeneity of application state and the execution environment.

This chapter presents the Accord autonomic services architecture that addresses these requirements and enables self-managing Grid applications. Accord extends the service-based Grid programming paradigm to relax static (defined at the time of instantiation) application requirements and system/application behaviors and allow them to be dynamically specified using high-level rules. Further, it enables the behaviors of services and applications to be sensitive to the dynamic state of the system and the changing requirements of the application and to adapt to these changes at runtime. This is achieved by extending Grid services to include the specifications of policies (in the form of high-level rules) and mechanisms for self-management, and providing a decentralized runtime infrastructure for consistently and efficiently enforcing these policies to enable autonomic self-managing functional, interaction, and composition behaviors based on current requirements, state and execution context. The design and implementation of Accord is presented. Accord is part of Project AutoMate [69], which provides the underlying middleware services.

This chapter also describes the use of Accord to enable the adaptive transfer of multi-terabyte data from live simulations running on supercomputers at NERSC and ORNL to local visualization and analysis clusters at PPPL while minimizing overheads to the simulation.

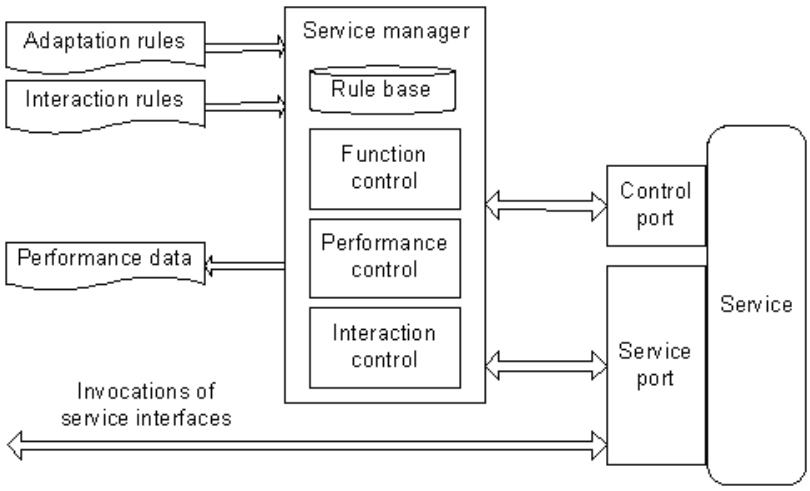

Figure 5.1: An Autonomic Service in Accord

The rest of the chapter is organized as follows. Section 5.1 describes the design and implementation of the Accord autonomic service architecture. Section 5.2 illustrates self-managing behaviors enabled by Accord using the data streaming application. Section 5.3 presents a conclusion.

5.1 Mechanisms for Self-Management using Rules

The Accord programming framework defines conceptual, implementation and enforcement models for utilizing human knowledge (in the form of rules) to guide the execution and adaptation of services. This is achieved by adapting the behaviors of individual services and their interactions (communication and coordination) to changing application requirements/state and execution environments based on dynamically defined rules.

5.1.1 Definition of Self-Managing Services

An autonomic (self-managing) service (see Figure 5.1) extends a Grid service with a control port for external monitoring and steering, and a service manager that monitors and controls the runtime behaviours of the managed element/service. The control

port consists of sensors that enable the state of the service to be queried, and actuators that enable the behaviours of the service to be modified. The control port and service port are used by the service manager to control the functions, performance, and interactions of the managed service. The control port is described using WSDL(Web Service Definition Language) [29] and may be a part of the general service description, or may be a separate document to control access to it. An example of the control port is shown in Table 5.1. Rules are simple if-condition-then-action statements described using XML and include service adaptation and service interaction rules. An example of a rule is shown in Table 5.2.

5.1.2 The Runtime Infrastructure

The Accord runtime infrastructure (shown in Figure 5.2) consists of a user/developer portal, peer service and application composition or coordination managers, the autonomic services, and a decentralized rule enforcement engine. This infrastructure enables adaptations of the behaviours of individual services as well as the interactions between services.

Behaviour Adaptation: Behaviour adaptation rules are used to adapt the behaviours of individual services and do not change their functionalities (described by service ports as contracts) and as a result, these adaptations are transparent to other services. This localized adaptation simplifies the specification and execution of adaptation rules by restricting the conditions monitored and actions performed within the individual services.

Behaviour adaptations include modification of service parameters and dynamic selection of algorithms and implementations to optimize and tune service performance, meet QoS requirements, correct detected errors, avoid or recover from failures, and/or to protect the service. Service managers execute these rules to adapt the functional behaviours of the managed services, and evaluate and tune their performance. These adaptations are realized by invoking appropriate control (sensors, actuators) and

functional interfaces.

Interaction Adaptation: An application composition manager decomposes incoming application workflows (defined by the user or a workflow engine) into interaction rules for individual services, and forwards these rules to corresponding service managers. Service managers execute these rules to establish interaction relationships among services by negotiating communication protocols and mechanisms and dynamically constructing coordination relationships in a distributed and decentralized manner.

Interaction rules are used to adapt service interactions, for example communication paradigms and/or coordination relationships. When local optimization of individual services cannot satisfy the global objectives, interaction rules are used to modify the application composition.

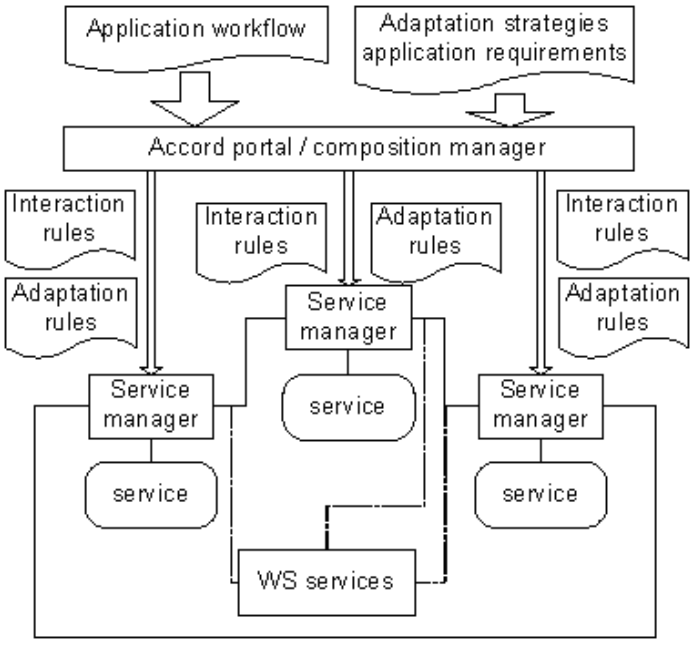

Figure 5.2: Accord Runtime Infrastructure: Solid Lines indicate Interactions among Services and Dotted Lines represent Invocation of WS Instances Providing Supporting Services such as Naming and Discovery

Rule Execution: Rule execution at the service managers consists of three phases:

condition inquiry, condition evaluation and conflict resolution, and batch action invocation. During condition inquiry, the service managers query the sensors used by the rules in parallel, assimilates their current values and fire corresponding triggers.

During the next phase, condition evaluations for all the rules are performed in parallel. Rule conflicts are detected during this phase when the same actuator is invoked with different values. These conflicts are resolved by relaxing the rule condition, using user-defined strategies, until the actuator-actuator conflict is resolved. If the conflicts are not resolved, errors are reported to users. If interacting services try to use different communication/coordination paradigms as a result of their independent adaptation behaviours, the services negotiate with each other to resolve the conflict [53].

After rule conflict resolution, the actions are executed in parallel. Note that the rule execution model presented here focuses on correct and efficient execution of rules, providing mechanisms to detect and resolve conflicts at runtime. However, correctness of rules and conflict resolution strategies are the responsibilities of the users. Rules

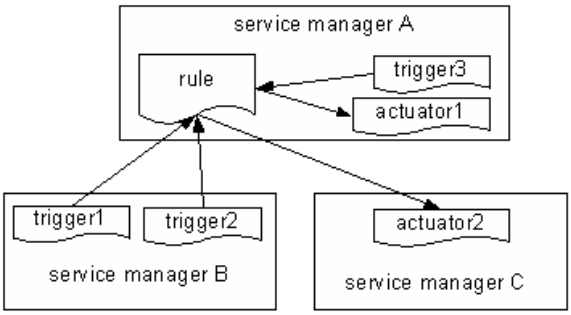

Figure 5.3: Execution of a Simple Rule in Accord

are evaluated and executed by service managers as shown in Figure 5.3. In the figure, the condition part of the sample rule consists of three *triggers* belonging to service A and B, and the action part has two actions that invoke the *actuators* exposed by service A and C. Triggers are injected into corresponding service managers A and B, and their results are collected by the service manager A. Service manager A evaluates the condition, invokes *actuator1* and notifies service manager C to invoke *actuator2*.

5.1.3 Autonomic Service Adaptation and Composition

Dynamic and autonomic compositions are enabled in Accord using a combination of interaction and adaptation rules. Composition consists of defining the organization of services and the interactions among them [53]. The service organization describes a collection of services that are functionally compose-able, determined semantically (e.g., using OWL(Ontology Workflow Language) [88]) or syntactically using WSDL [29]. Interactions among services define the coordination between services and the communication paradigm used, e.g., message passing, RPC/RMI, or shared spaces.

Once a workflow has been generated (e.g., using the mechanism in [5]), and the services have been discovered (using middleware services), the Accord composition manager decomposes the workflow into interaction rules. This decomposition process consists of mapping workflow patterns [84] in the workflow into corresponding rule templates [53]. Accord provides templates for basic communication paradigms such as notification, publisher/subscriber, rendezvous, shared spaces and RPC/RMI, and control structures such as sequence, AND-split, XOR-split, OR-split, AND-join, XOR-join, and OR-join. More complex interaction and coordination structures (e.g., loops) can be constructed from these basic patterns.

The interaction rules are then injected into corresponding service managers, which execute the rules to establish communication and coordination relationships among involved services. Note that there is no centrally controlled orchestration. While the interaction rules are defined by the composition manager, the actual interactions are established by service managers in a decentralized and parallel manner.

The communication paradigms and coordination relationships among the interacting autonomic services can be dynamically changed according to current application state and execution context by replacing or changing the related interaction rules. As a result, a new service can be brought into an application, and interactions among services can be changed at runtime, without taking the application offline.

The two adaptation approaches, adaptation within individual services and dynamic composition of services, can be used separately or in combination to enable the autonomic self-configuring, self-optimizing and self-healing behaviours of services and applications [53].

5.1.4 Implementation Overview

The prototype implementation of the Accord autonomic services architecture extends the Apache Axis Toolkit and is being integrated with the Globus toolkit GT4 In our current version, both control ports and service ports are implemented as WSDL documents. Service ports are invoked by interacting services, and control ports are used by managers to periodically querying and modifying service behaviors. The publication/subscription structure is used between managers. Each manager maintains a subscription tables and publishes trigger information to subscriber managers using XML messages.

Further, it uses middleware services provided by AutoMate [69] to enable (1) content-based routing/discovery, associative messaging, and a decentralized reactive tuple space for interactions among service managers, and (2) context-based access control and cooperative protection for service authorization and authentication. An experimental evaluation of Accord and its overheads are presented in [53].

5.2 Self-Managing Data Streaming using Accord

5.2.1 Application Setup

This section illustrates the self-managing behaviors enabled by the Accord service architecture using an autonomic data streaming service. The overall application is presented in Figure 5.4. The application consists of the G.T.C. fusion simulation

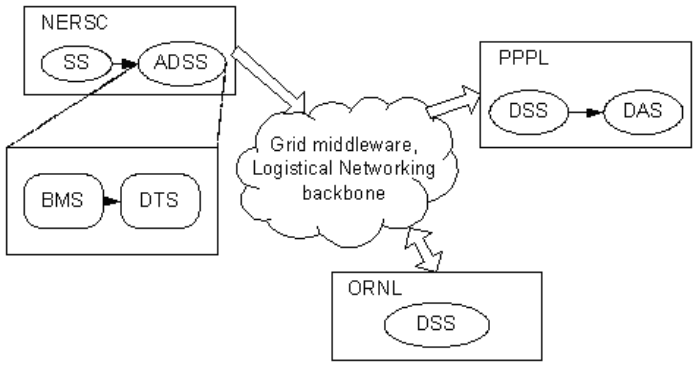

Figure 5.4: The Self Managing Data Streaming Service

that runs for days on a parallel supercomputer at NERSC (CA) and generates multi-terabytes of data. This data is analyzed and visualized live, while the simulation is running, at PPPL (NJ). The data also has to be archived either at PPPL (NJ) or ORNL (TN). Data streaming techniques from a large number of processors have been shown to be more beneficial for such a runtime analysis than writing data to the disk in the previous section [15, 47]. The goal of the autonomic data steaming service is to stream data from the live simulation to support remote runtime analysis and visualization at PPPL while minimizing overheads on the simulation, adapting to network conditions, and eliminating loss of data. The application workflow consists of following five core services:

- The **Simulation Service (SS)** executes in parallel on 6K processors of the Seaborg IBM SP machine at NERSC and generates data at regular intervals that has to be transferred at runtime for analysis and visualization at PPPL, and archived at data stores at PPPL or ORNL.

- The **Data Analysis Service (DAS)** runs on a 32 node cluster located at PPPL. The service analyzes and visualizes the steaming data.

- The **Data Storage Service (DSS)** archives the streamed data using the Logistical Networking backbone [73], which builds a Data Grid of storage services

located at ORNL and PPPL.

- The **Autonomic Data Streaming Service (ADSS)** is constructed using the Accord autonomic services architecture and manages the streaming of data from the simulation service to the DAS (at PPPL) and DSS (at PPPL/ORNL). It is a composite service composed of two services:

  - The **Buffer Manager Service (BMS)** manages the buffers allocated by the service based on the rate and volume of data generated by the simulation and determines the granularity of blocks used for data transfer.
  - **Data Transfer Service (DTS)** manages the transfer of blocks of data from the buffers to remote services for analysis and visualization at PPPL, and archiving at PPPL or ORNL. The transfer service uses the IBP [71] protocol to transfer data

As mentioned above, the objective of ADSS is to minimize overheads of data transfer on the simulation, adapt the transfer to network conditions, and ensure that there is no loss of data. Three self-managing scenarios for ADSS are described below.

5.2.2 Self-Managing Scenarios using Rule based Adaptations

Scenario 1: Self-optimizing behaviour of BMS

This scenario illustrates the self-optimizing behaviour of the BMS using rules. The service adaptation within BMS service is transparent to other services. BMS selects the appropriate blocking technique, orders blocks in the buffer and optimizes the size of the buffer(s) used to ensure low latency high performance steaming and minimize the impact on the execution of the simulation. The adaptations are based on the current state of the simulation and more specifically the following three runtime parameters. (1) The data generation rate, which is the amount of data generated per iteration divided by the time required for the iteration, and can vary from 1 to

400 Mbps depending on the domain decomposition and the type of analysis to be performed. (2) The network connectivity and the network transfer rate. The latter is limited by the 100 Mbps link between NERC and PPPL. (3) The nature of data being generated in the simulation, e.g., parameters, 2D surface data or 3D volume data. BMS provides three algorithms:

- Uniform Buffer Management: This algorithm divides the data into blocks of fixed sizes, which are then transmitted by the DTS. This static algorithm is more suited for the simulations generating data at a small or medium rate (50Mbps). Using smaller block sizes have significant advantages at the receiving end as less time is required for decoding the data and processing it for analysis and visualization.

- Aggregate Buffer Management: This algorithm aggregates blocks across iterations and the DTS transmits these aggregated blocks. This algorithm is suited for high data generation rates, i.e., between 60-400 Mbps.

- Priority Buffer Management: This algorithms orders data blocks in the buffer based on the nature of the data. For example, 2D data blocks containing visualization or simulation parameters are given higher priority as compared to 3D raw volume data. To enable adaptations, the BMS exports two sensors, "DataGenerationRate" and "DataType", and one actuator, "BlockingAlgorithm" as part of its control port shown in Table 5.1. This document describes the name, type, message format and protocol details for each sensor/actuator. Further, the BMS self-optimization behaviour is governed by the rule shown in Table 5.2, which states that if the data generation rate is greater than the peak network transfer rate (i.e., 100 Mps), the aggregate buffer management is used otherwise the uniform buffer management algorithm is used.

The resulting adaptation behaviour is plotted in Figure 5.5. The figure shows that BMS switches to aggregate buffer management during simulation time intervals 75

Table 5.1: The Control Port for the BMS

```
<controlPort name="BMS_controlPort" service="BufferManagerService">
   <types>
      <sensor name="DataGenerationRate">
         <element name="DataGenerationRateReq" type="string"/>
         <element name="DataGenerationRateResp" type="double"/>
      </sensor>
      <sensor name="DataType">
         <element name="DataTypeReq" type="string"/>
         <element name="DataTypeResp" type="string"/>
      </sensor>
      <actuator = name="BlockingAlgorithm">
         <element name="BlockingAlgorithmReq" type="string"/>
      </actuator>
   </types>
   <message name="GetDataGenerationRateIn">
      <part name="body" element="DataGenerationRateReq"/>
   </message>
   <message name="GetDataGenerationRateOut">
      <part name="body" element="DataGenerationRateResp"/>
   </message>
   <message name="GetDataTypeIn">
      <part name="body" element="DataTypeReq"/>
   </message>
   <message name="GetDataTypeOut">
      <part name="body" element="DataTypeResp"/>
   </message>
   <message name="SetBlockingAlgorithm">
      <part name="body" element="BlockingAlgorithmReq"/>
   </message>
   <portType name="BMSControlPortType">
      <operation name="SensorDataGenerationRate">
         <input message="tns:GetDataGenerationRateIn"/>
         <output message="tns:GetDataGenerationRateOut"/>
      </operation>
      <operation name="SensorDataType">
         <input message="tns:GetDataTypeIn"/>
         <output message="tns:GetDataTypeOut"/>
      </operation>
      <operation name="ActuatorBlockingAlgorithm">
         <input message="tns:SetBlockingAlgorithm"/>
      </operation>
   </portType>
</controlPort>
```

Table 5.2: The Adaptation Rule for the BMS

```
<rule name="BlockingRule" attribute="active">
   <trigger name="2D" sensor="DataType" op="EQ" value="2D" type="string"/>
   <trigger name="DGR" sensor="DataGenerationRate" op="GT" value=peakRate type="float"/>
   <when>
      <and>
         <operand trigger="2D"/>
         <operand trigger="DGR"/>
      </and>
   </when>
   <do>
      <action actuator="BlockingAlgorithm">
         <input value="priorityAggregation" type="string"/>
      </action>
   </do>
   <when>
      <and>
         <operand trigger="2D"/>
         <not>
            <operand trigger="DGR"/>
         </not>
      </and>
   </when>
   <do>
      <action actuator="BlockingAlgorithm">
         <input value="priority" type="string"/>
      </action>
   </do>
   <when>
      <and>
         <operand trigger="DGR"/>
         <not>
            <operand trigger="2D"/>
         </not>
      </and>
   </when>
   <do>
      <action actuator="BlockingAlgorithm">
         <input value="aggregate" type="string"/>
      </action>
   </do>
   <else>
      <action actuator="BlockingAlgorithm">
         <input value="uniform" type="string"/>
      </action>
   </else>
</rule>
```

sec to 150 sec and 175 sec to 250 sec, as the simulation data generation rate peaks to 100Mbps and 120 Mbps during these intervals. The aggregation is an average of 7 blocks. Once the data generation rate falls to 50Mbps, BMS switches back to the uniform buffer management scheme, and constantly sends 3 blocks of data on the network. Figure 5.6 plots the percentage overhead on the simulation execution with and without autonomic management (using rules). Overhead is computed as the absolute difference between the time required to generate data without the ADSS service and the time required to stream the data using ADSS service.

The plot shows that the BMS switches from uniform buffer management to aggregate buffer management at data generation rates of around 80-90 Mbps. This increases the overhead slightly, however the overheads remains less than 5%. Without autonomic management, the overheads increase to about 10% for higher data rates as the BMS continues to use uniform buffer management.

When the simulation service generates 2D visualization data in addition to 3D data, the priority buffer management algorithm is triggered. The 2D data blocks are given higher priority and are moved to the head to data transmission queue. As a result, transmission of the 2D data is expedited with almost no impact to the 3D data.

Scenario 2: Self-configuring/self-optimizing behaviour of the ADSS

The effectiveness of the data transfer between the simulation service at NERSC and the analysis or visualization service at PPPL depends on the network transfer rate, which depends on data generation rates and/or network conditions. Falling network transfer rates can lead to buffer overflows and require the simulation to be throttled to avoid data loss. One option to maintain data throughputs is to use multiple data streams. Of course, this option requires multiple buffers and hence uses more of the available memory. Implementing this option requires the creation of multiple instances of ADSS. In this scenario, ADSS monitors the effective network transfer rate, and when this rate dips below a certain threshold, the service causes another

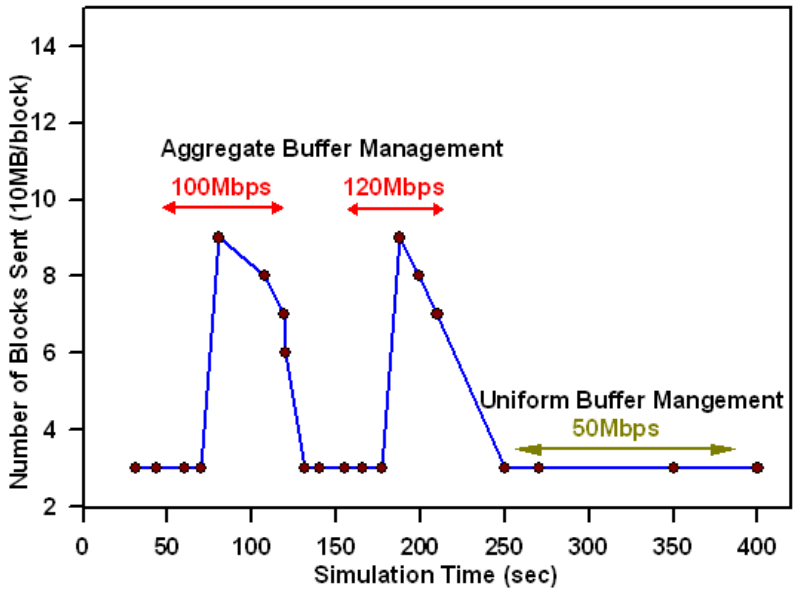

Figure 5.5: Self-Optimization Behaviour of the Buffer Management Service (BMS) - BMS Switches Between Uniform and Aggregate Blocking Algorithms based on Data Generation Rates, Network Transfer Rates and the Nature of Data Generated

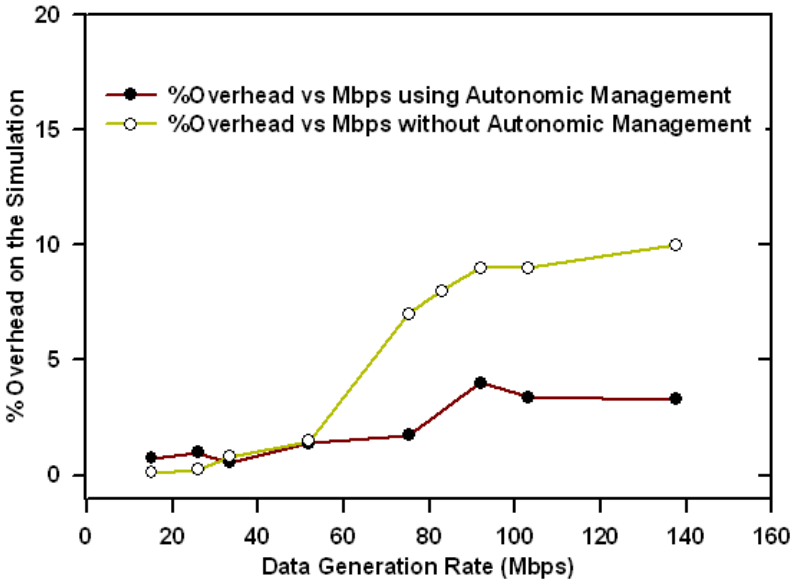

Figure 5.6: Percentage Overhead on Simulation Execution With and Without Autonomic Management using Rules

instance of the ADSS to be created and incorporated into the workflow. Note that the maximum number of ADSS instances possible is predefined. Similarly, if the effective data transfer rate is above a threshold, the number of ADSS instances is decreased to reduce memory overheads. The upper and lower thresholds have been determined using experiments in [15]. The self-configuration behaviour of ADSS is governed by

Table 5.3: The Self-Configuring Rule for the ADSS

```
<rule name="SplitRule" attribute="active">
   <trigger name="SmallNTR" sensor="NetworkTransferRate"
      op="LT" value=lowerthreshold type="float"/>
   <trigger name="LargeNTR" sensor="NetworkTransferRate"
      op="GT" value=upperthreshold type="float"/>
   <trigger name="ADSSNum" sensor="NumOfADSS"
      op="LT" value=num type="integer"/>
   <when>
      <and>
         <operand trigger="SmallNTR"/>
         <operand trigger="ADSSNum"/>
      </and>
   </when>
   <do>
      <action actuator="Accord:NewInstances">
         <input value="BMS" type="service"/>
      </action>
      <action actuator="Accord:LoadRules">
         <input value="BMS" type="service"/>
         <input value="BMSRuleName" type="string"/>
      </action>
      <action actuator="Accord:NewInstances">
         <input value="DTS" type="service"/>
      </action>
      <action actuator="Accord:LoadRules">
         <input value="DTS" type="service"/>
         <input value="DTSRuleName" type="string"/>
      </action>
   </do>
   <when>
      <operand trigger="LargeNTR"/>
   </when>
   <do>
      <action actuator="Accord:GetInstances">
         <input value="BMS" type="service"/>
         <output value="BMSInstanceList"
            type="serviceInstanceList"/>
      </action>
      <action actuator="Accord:DelInstances">
         <input value="BMSInstanceList"
            type="serviceInstanceList"/>
         <input value="number" type="integer"/>
      </action>
   </do>
</rule>
```

the rule shown in Table 5.3. When the network transfer rate is below a pre-defined threshold, ADSS will use Accord to create new instances of ADSS including BMS and

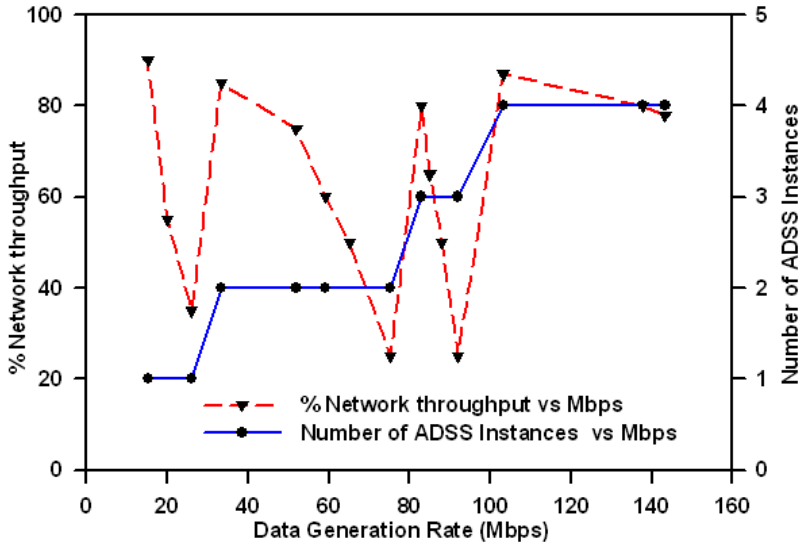

Figure 5.7: Effect of Creating New Instances of the ADSS Service when the %Network Throughput Dips Below the User Defined (50%) Threshold

DTS and load corresponding rules into the new BMS and DTS instances to enable interactions between them. When the network transfer rate is above a pre-defined threshold, ADSS obtains the list of exiting ADSS instances using the Accord runtime, and deletes a pre-defined number of instances. The resulting behaviours are plotted in Figure 5.7. This figure plots the percentage of network throughput, which is the difference between the current network transfer rate and the maximum network rate between PPPL and NERSC, i.e., 100 Mbps. The figure shows that the number of ADSS instances first increases as the network throughput dips below the 50% threshold (corresponding to data generation rates of around 25 Mbps in the plot), as defined by the rule in Table 5.3. This causes the network throughput to increase to above 80%. Even more instances of ADSS services are created at data generation rates of around 40 Mbps and the network throughput once again jumps to around 80Mbps. The ADSS instances increase until the limit of 4 is reached.

Scenario 3: Self-healing behaviour of the ADSS

This scenario addresses data loss in the cases of extreme network congestion or network failures. These cases cannot be addressed using simple buffer management or

Table 5.4: The Self-Healing Rule for the ADSS

```
<rule name="TransferRule" attribute="active">
   <trigger name="transferFailed" sensor="DataTransfer"
      op="EQ" value="0" type="integer"/>
   <trigger name="transferSwitch" sensor="NumOfSwitches"
      op="LT" value=switchThreshold type="integer"/>
   <when>
      <and>
         <operand trigger="transferFailed"/>
         <operand trigger="transferSwitch"/>
      </and>
   </when>
   <do>
      <action actuator="TransferAlgorithm">
         <input value="remote" type="string"/>
      </action>
   </do>
   <when>
      <not>
         <operand trigger="transferSwitch"/>
      </not>
   </when>
   <do>
      <action actuator="TransferAlgorithm">
         <input value="remote" type="string"/>
      </action>
      <action actuator="Accord:SetRuleAttribute">
         <input value="TransferRule" type="string"/>
         <input value="inactive" type="string"/>
      </action>
   </do>
</rule>
```

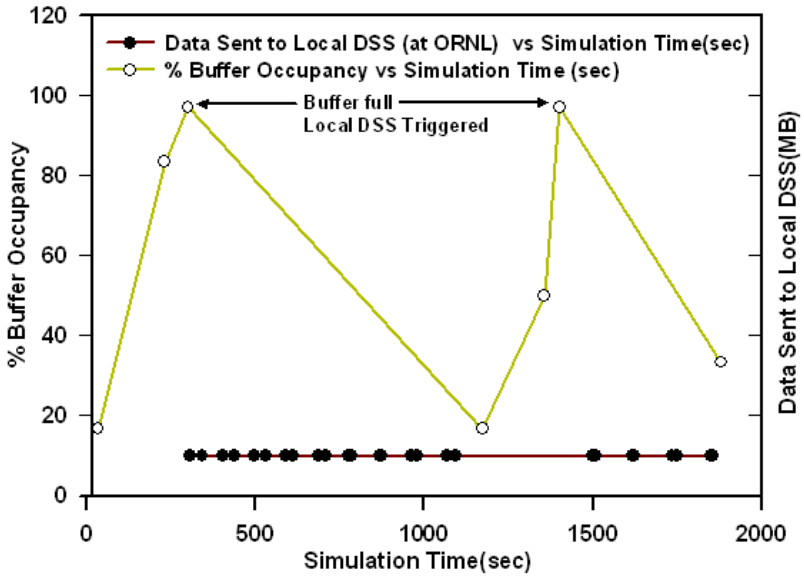

Figure 5.8: Effect of Switching from the DSS at PPPL to the DSS ORNL in Response to Network Congestion and/or Failure

replication. One option in these cases to avoid loss of data is to write data locally at NERSC rather than streaming. However, this data will not be available for analysis and visualization until the simulation complete, which could be days. Writing data to the disk also causes significant overheads to the simulation [15]. ADSS addresses these cases by temporarily or permanently switching the streaming of the data to the DSS at ORNL instead of PPPL. NERSC and ORNL are connected by a low latency [50] link which has a lower probability of being saturated. The data can be later transmitted from ORNL to PPPL. Congestion is detected by observing the buffer - when the buffer is filled to a capacity, the ADSS switches subsequent streaming to ORNL, and when the buffer is no longer saturated, switches the steaming back to PPPL. If the service observes that buffer is being continuously saturated, it infers that there is a network failure and permanently switches the streaming to ORNL. In this case, the blocks already in the PPPL buffer are transferred to the ORNL queue. Here ADSS communicates with DSS at PPPL or DSS at ORNL under different network conditions. This behaviour is defined by interaction rules in ADSS. The rule specifying this self-management behaviour is listed in Table 5.4. The resulting self-healing

behaviour is plotted in Figure 5.8. The figure shows that as the ADSS buffer(s) get saturated, the data streaming switches to the DSS at ORNL, and when the buffer occupancy falls below 20% it switches back to PPPL. Note that while the data blocks are written to ORNL, data blocks already queued for transmission to PPPL continue to be streamed. The figure also shows that, at simulation time 1500 (X axis), the PPPL buffers once again get saturated and the streaming switches to ORNL. If this persists, the steaming would be permanently switched to ORNL.

5.3 Conclusions

This chapter presented the Accord services architecture for self-managing Grid applications. It enables the development of self-managing services and the formulation of self-managing applications as the dynamic composition of these services, where the runtime computational behavior of the services as well as their compositions and interactions can be managed at runtime using dynamically injected rules. As a result, applications are capable of adapting their runtime behaviors to deal with the dynamism and uncertainty of the nature of Grids and Grid applications. An self-managing data streaming application was used to illustrate the self-managing behaviors enabled by this software framework. As platforms change and software evolves, the rules may need to be changed and thresholds need to be modified. Using this approach, rule maintenance in the self-managing data streaming approach was performed manually. Advising these rules and automatically deriving thresholds [24] will be explored in detail in the next chapter.

Chapter 6

Self-Managing Data Streaming using Model based Online Control

This chapter presents the design, implementation and experimental evaluation of a self-managing data streaming service for wide-area Grid environments. The service is deployed using an infrastructure for self-managing Grid services, including a programming system for specifying self-managing behaviour as well as models and mechanisms for enforcing this behaviour at runtime [18]. A key contribution of this chapter is the combination of typical rule-based self-management approaches with formal model-based online control strategies. While the former are relatively simple and easy to implement, they require a great deal of expert knowledge, are very tightly coupled to specific applications and their performance is difficult to analyse in terms of optimality, feasibility and stability properties. Advanced control formulations offer a theoretical basis for self-managing adaptations in distributed applications. Specifically, this chapter combines model-based limited look-ahead controllers (LLC) with rule-based managers to dynamically achieve adaptive behaviour in Grid applications under various operating conditions [54].

This chapter demonstrates the operation of the proposed data streaming service using a Grid-based fusion simulation workflow consisting of long-running coupled simulations, executing on remote supercomputing sites at NERSC (National Energy Research Scientific Computing Center) in California (CA) and ORNL (Oak Ridge National Laboratory) in Tennessee (TN) and generating several terabytes of data, which must be streamed over the network for live analysis and visualization at PPPL (Princeton Plasma Physics Laboratory) in New Jersey (NJ) and for archiving at

ORNL (TN). The service aims to minimize the overhead associated with data streaming on the simulation, adapt quickly to network conditions and prevent any loss of simulation data.

The rest of this chapter is organized as follows. Section 6.1 describes the models and mechanism for enabling self-managing Grid services and applications. Section 6.2 presents the design, implementation, operation and evaluation of the self-managing data streaming service. Section 6.3 addresses the scalability of the service and proposes and evaluates hierarchical control strategies. Section 6.4 concludes the chapter.

6.1 Model and Mechanisms for Self-Management

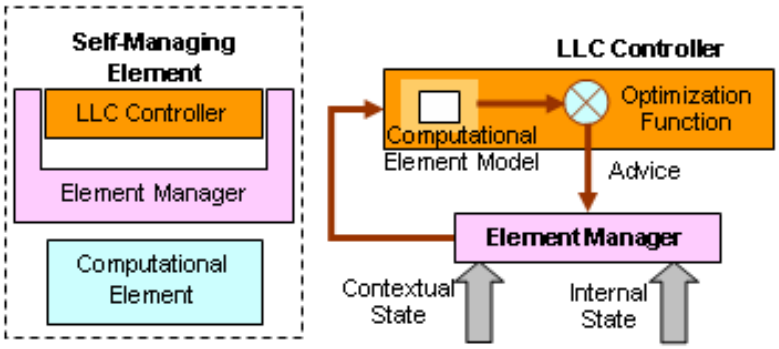

Figure 6.1: A Self Managing Element and Interactions between the Element Manager and Local Controller.

The data streaming service presented in this chapter is constructed using the Accord infrastructure [18], [54], which provides the core models and mechanisms for realizing self managing Grid services. Its key components are shown in Figure 6.1 and are described in the following sections.

6.1.1 A Programming System for Self-Managing Services

The programming system extends the service-based Grid programming paradigm to relax assumptions of static (defined at the time of instantiation) application requirements and system/application behaviors and allows them to be dynamically specified

using high-level rules. Further, it enables the behaviors of services and applications to be sensitive to the dynamic state of the system and the changing requirements of the application and to adapt to these changes at runtime. This is achieved by extending Grid services to include the specifications of policies (in the form of rules) and mechanisms for self-management and providing a decentralized runtime infrastructure for consistently and efficiently enforcing these policies to enable self-managing functional, interaction and composition behaviors based on current requirements, state and execution context.

A self-managing service extends a Grid service with a control port for external monitoring and steering. An element manager monitors and controls the runtime behaviors of the managed service/element according to changing requirements and state of applications as well as their execution environment. The control port consists of sensors and actuators, which may be parameters, variables, or functions and enable the state of the service to be queried and the behaviors of the service to be modified. The control port and service port are used by the service manager to control the functions, performance and interactions of the managed service. The control port is described using WSDL (Web Service Definition Language) [29] and may be a part of the general service description, or may be a separate document with access control. Polices are in the form of simple if-condition then-action rules described using XML and include service adaptation and service interaction rules. Examples of control ports and policy specifications can be found in [54].

6.1.2 Online Control Concepts

Figure 6.2 shows the overall LLC framework [1], [44], where the management problem is posed as a sequential optimization under uncertainty. Relevant parameters of the operating environment (such as data generation patterns and effective network bandwidth) are estimated and used by a mathematical model to forecast future application behavior over a prediction horizon N. The controller optimizes the forecast

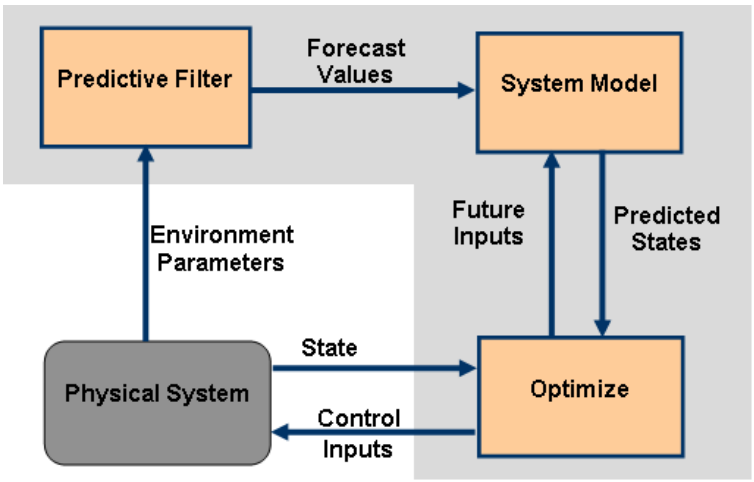

Figure 6.2: The LLC Control Structure

behavior as per the specified QoS requirements by selecting the best control inputs to apply to the system. At each time step k, the controller finds a feasible sequence $u^*(i)|i \in [k+1, k+N]$ of inputs (or decisions) within the prediction horizon. Then, only the first move is applied to the system and the whole optimization procedure is repeated at time k +1 when the new system state is available.

The LLC approach allows for multiple QoS goals and operating constraints to be represented in the optimization problem and solved for each control step. It can be used as a management scheme for systems and applications where control or tuning inputs must be chosen from a finite set and those exhibiting both simple and nonlinear dynamics. In addition, it can accommodate run-time modifications to the system model itself caused by resource failures, dynamic data injection and time-varying parameter changes. The following discrete-time state-space equation describes the system dynamics.

$$x(k+1) = f(x(k), u(k), \omega(k))$$

where $x(k) \in \Re^k$ is the system state at time step k and $u(k) \in U \subset \Re^m$ and $\omega(k) \in \Re^r$ denote the control inputs and environment parameters at time k, respectively. The

$$\begin{aligned}
&\textbf{Minimize} \quad \sum_{i=k+1}^{k+N} J\left(x(i), u(i)\right) \\
&\textbf{Subject to :} \\
&\hat{x}(i+1) = f\left(x(i), u(i), \hat{\omega}(i)\right), \\
&H\left(x(i)\right) \leq 0, \\
&u(i) \subset U\left(x(i)\right)
\end{aligned}$$

Figure 6.3: The Look-Ahead Optimization Problem

system dynamics model f captures the relationship between the observed system parameters, particularly those relevant to the QoS specifications and the control inputs that adjust these parameters.

Though environment parameters such as workload patterns in Grid environments are typically uncontrollable, they can be estimated online with some bounded error using appropriate forecasting techniques, for example, a Kalman filter [45]. Since the current values of the environment inputs cannot be measured until the next sampling instant, the corresponding system state can only be estimated as:

$$\hat{x}(k+1) = f(x(k), u(k), \hat{\omega}(k))$$

where $\hat{x}(k+1)$ is the estimated system state and $\hat{\omega}(k)$ denotes the environment parameters estimated by the forecasting model(s). A self-managing application must achieve specific QoS objectives while satisfying its operating constraints. These objectives may be expressed as a *set-point specification* where the controller aims to operate the system close to the desired state $x^* \in X$ where X is the set of valid system states. The application must also operate within strict constraints on both the system variables and control inputs. A general form is used to describe the operating constraints of interest as $H(x(k)) \leq 0$ while $u(x(k)) \subseteq U$ denotes the control-input set $u(x(k))$ permitted in state $x(k)$. It is also possible to consider *transient* or *control costs* as part of the system operating requirements, indicating that certain trajectories

towards the desired state are more preferable over others in terms of their cost to the system. The overall performance specification will then require that the system reach its setpoint while minimizing the corresponding control costs. This specification is captured by the following norm-based function J that defines the overall operating cost at time k.

$$J(x(k), u(k)) = \| x(k) - x^* \|_P + \| u(k) \|_Q + \| \Delta u(k) \|_R$$

where $\Delta u(k) = u(k) - u(k-1)$ is the change in control inputs; P, Q and R are user-defined weights denoting the relative importance of the variables in the cost function. The optimization problem of interest is then posed in Figure 6.3 and solved using the LLC structure introduced in Figure 6.2.

6.1.3 Operation

The element (service) managers provided by the programming system are augmented with controllers, allowing them to use model-based control and optimization strategies [18]. A manager monitors the state of its underlying elements and their execution context, collects and reports runtime information and enforces adaptation actions determined by its controller. The enhanced managers thus augment human-defined rules, which may be error-prone and incomplete, with mathematically sound models, optimization techniques and runtime information. Specifically, the controller decides when and how to adapt the application behavior and the managers focus on enforcing these adaptations in a consistent and efficient manner.

6.2 The Self Managing Data Streaming Service

This section describes a self-managing data streaming services to support a Grid-based fusion simulation, based on the models and mechanisms presented in the previous

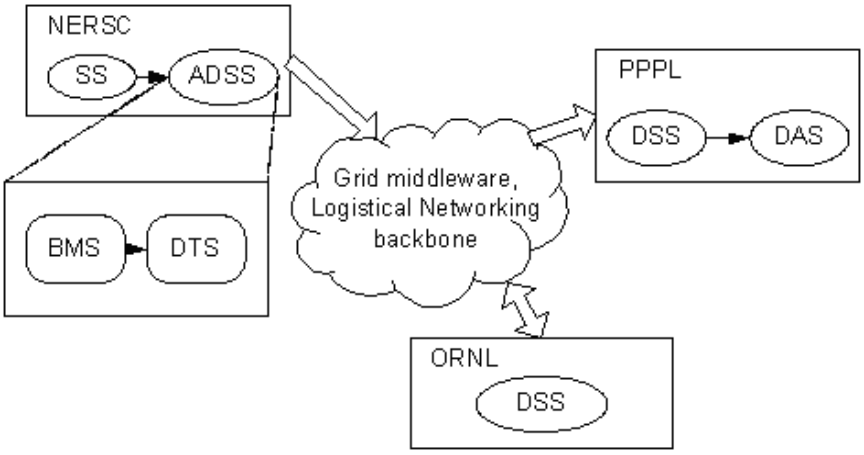

Figure 6.4: The Self Managing Data Streaming Application

section. A specific driving simulation workflow is shown in Figure 6.4 and consists of a long running G.T.C. fusion simulation executing on a parallel supercomputer at NERSC (CA) and generating terabytes of data over its lifetime. This data must be analyzed and visualized in real time, while the simulation is still running, at a remote site at PPPL (NJ) and also archived either at PPPL (NJ) or ORNL (TN). Data streaming techniques from a large number of processors have been shown to be more beneficial for such a runtime analysis than writing data to the disk [47].

The data streaming service in Figure 6.4 is composed of 4 core services:

1. A *Simulation Service (SS)* executing on an IBM SP machine at NERSC and generating data at regular intervals that has to be transferred at runtime for analysis and visualization at PPPL and archived at data stores at PPPL or ORNL.

2. A *Data Analysis Service (DAS)* executing on a computer cluster located at PPPL to analyze the data streamed from NERSC.

3. A *Data Storage Service (DSS)* to archive the streamed data using the Logistical Networking backbone [73], which builds a Data Grid of storage services located at ORNL and PPPL.

4. An *Autonomic Data Streaming Service (ADSS)* that manages the data transfer from SS (at NERSC) to DAS (at PPPL) and DSS (at PPPL/ORNL). It is a composite service composed of two services:

   (a) The *Buffer Manager Service (BMS)* manages the buffers allocated by the service based on the rate and volume of data generated by the simulation and determines the granularity of blocks used for data transfer.

   (b) The *Data Transfer Service (DTS)* manages the transfer of blocks of data from the buffers to remote services for analysis and visualization at PPPL and archiving at PPPL or ORNL. The data transfer service uses the Internet BackPlane Protocol(IBP) [71] to transfer data.

The objectives of the self-managing ADSS are the following:

1. *Prevent any loss of simulation data*: Since data continuously generated and the buffer sizes are limited, the local buffer at each data transfer node must be eventually emptied. Therefore, if the network link to the analysis cluster is congested, then data from the transfer nodes must be written to a local hard disk at NERSC itself.

2. *Minimize overhead on the simulation*: In addition to transferring the generated data, the transfer nodes must also perform useful computations related to the simulation. Therefore, the ADSS must minimize the computational and resource requirements of the data transfer process on these nodes;

3. *Maximize the utility of the transferred data*: It is desirable to transfer as much of the generated data as possible to the remote cluster for analysis and visualization. Storage on the local hard disk is an option only if the available network bandwidth is insufficient to accommodate the data generation rate and there is a danger of losing simulation data.

6.2.1 Design of the ADSS Controller

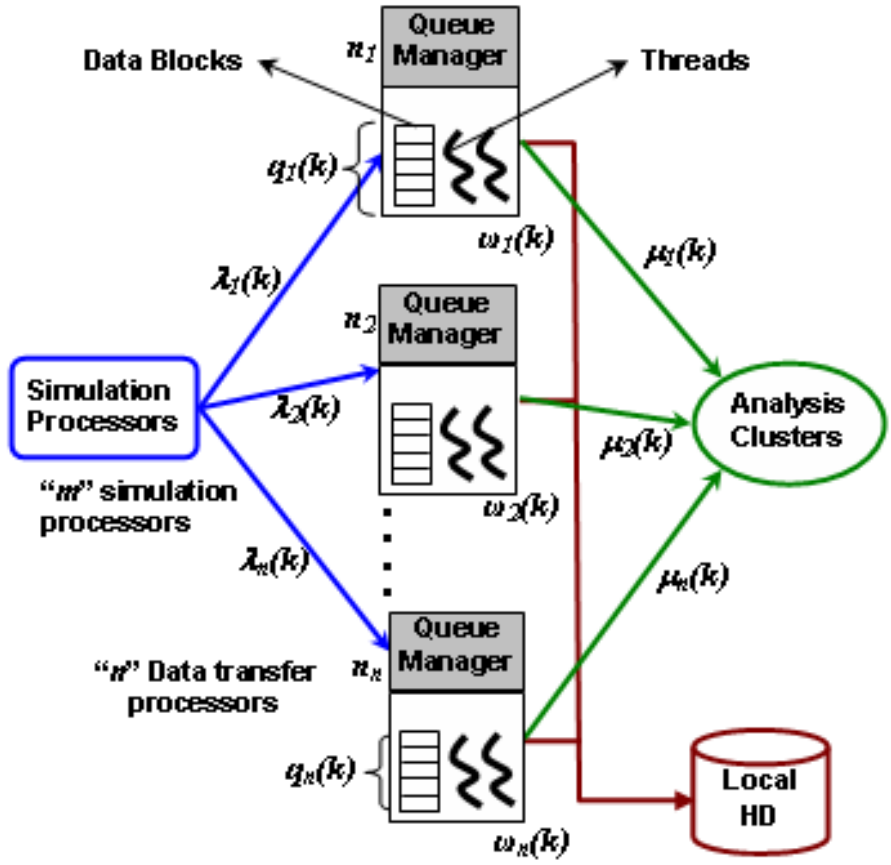

Figure 6.5: LLC Model for the ADSS controller

The ADSS controller is designed using the LLC concepts discussed in Section 6.1. Figure 6.5 shows the queuing model for the streaming service, where the key operating parameters for a data transfer node n_i at time step k are as follows: (1) State variable: The current average queue size at n_i denoted as $q_i(k)$; (2) Environment variables: $\lambda_i(k)$ denotes the data generation rate into the queue q_i and $B(k)$ the effective bandwidth of the network link; (3) Control or decision variables: Given the state and environment variables at time k, the controller decides $\mu_i(k)$ and $\omega_i(k)$, the data-transfer rate over the network link and to the hard disk respectively. The system dynamics at each node n_i evolves as per the following equations:

$$\hat{q}_i(k+1) = \hat{q}_i(k) + (\hat{\lambda}_i(k).(1 - \mu_i(k) - \omega_i(k))).T$$

$$\lambda_i(k) = \phi(\lambda_i(k-1), k)$$

The queue size at time $k+1$ is determined by the current queue size, the estimated data generation rate $\lambda_i(k)$ and the data transfer rates, as decided by the controller, to the network link and the local hard disk. The data generation rate is estimated using a forecasting model ϕ, implemented here by an exponentially-weighted moving-average (EWMA) filter. The sampling duration for the controller is denoted as T. Both $0 \leq \mu_i(k) \leq 1$ and $0 \leq \omega_i(k) \leq 1$ are chosen by the controller from a finite set of appropriately quantized values. Note that in practice, the data transfer rate is a function of the effective network bandwidth $B(k)$ at time k, the number of sending threads and the size of each data block transmitted from the queue. These parameters are decided by appropriate components within the data-streaming service (discussed in Section 6.2.2).

The LLC problem is now formulated as a set-point specification where the controller aims to maintain each node n_i's queue q_i around a desired value q^* while maximizing the utility of the transferred data, that is, by minimizing the amount of data transferred to the hard disk/local depots [71].

$$Minimize : \sum_{j=k}^{k+N} \sum_{i=1}^{n} \alpha_i (q^* - q_i(j))^2 + \beta_i \omega_i(j)^2$$

$$Subject\ to : \sum_{i=1}^{n} \mu_i(j) \leq B(j)\ and\ q_i(j) \leq q_{\texttt{max}}\ \forall i$$

Here, N denotes the prediction horizon, $q_{\texttt{max}}$ the maximum queue size and α_i and β_i denote user-specified weights in the cost function.

When control inputs must be chosen from a set of discrete values, the LLC formulation, as posed above, will show an exponential increase in worst-case complexity with an increasing number of control options and longer prediction horizons – the so called "curse of dimensionality". Since the execution time available for the controller is often limited by hard application bounds, it is necessary to consider the possibility

that it may have to deal with suboptimal solutions. For adaptation purposes, however, it is not critical to find the global optimum to ensure system stability that is; a feasible suboptimal solution will suffice. Taking advantage of the fact that the operating environment does not change drastically over a short period of time, suboptimal solutions are obtained using *local search methods*, where given the current values of $\mu_i(k)$ and $\omega_i(k)$, the controller searches a limited neighborhood of these values for a feasible solution for the next step.

6.2.2 Implementation and Deployment of ADSS

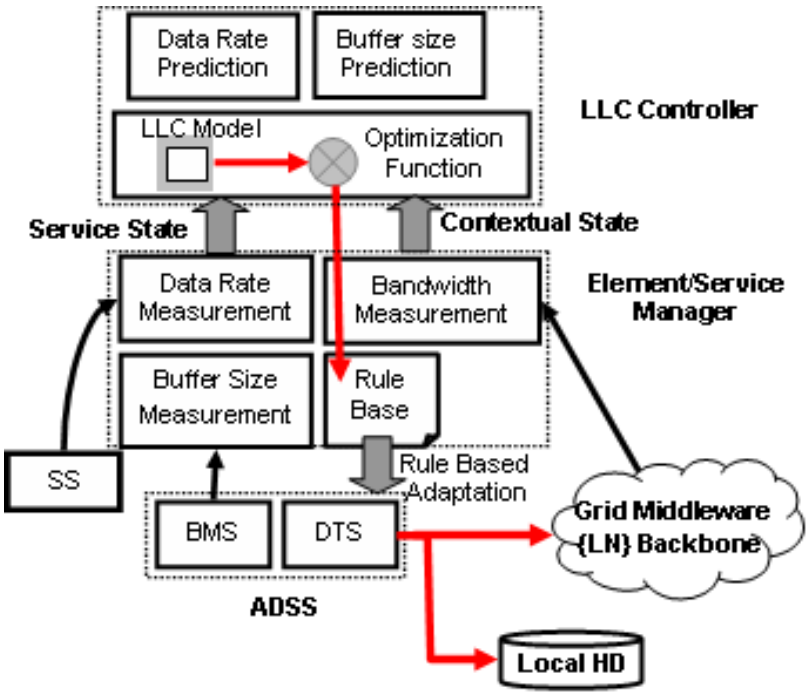

Figure 6.6: Implementation Overview of the ADSS

ADSS(refer to Figure 6.6) is implemented as a composite service comprising a *Buffer Manager Service* (BMS) that manages the buffers allocated by the ADSS and a *Data Transfer Service* (DTS) that manages the transfer of blocks of data from the buffers. The BMS supports two buffer management schemes, Uniform and Aggregate buffering. *Uniform buffering* divides the data into blocks of fixed sizes and is more

suitable when the simulation can transfer all its data items to a remote storage. *Aggregate buffering*, on the other hand, aggregates blocks across multiple time steps for network transfer and can be used when the network is congested. The control ports for these services are described in detail in [54].

The *ADSS Online Controller* consists of the system model, the set-point specification and the LLC scheme. The system model obtains inputs from the data generation rate prediction and buffer size prediction sub-module, which provides it with future values of the data rates (sizes) and future buffer capacities respectively. The prediction of the data generation rate uses a EWMA filter with a smoothing constant of 0.5. A single-step LLC scheme ($N = 1$) is implemented on each node/data transfer processor n_i with a desired queue size of $q^* = 0$. The weights in the multi objective cost function are set to $\alpha_i = 1$ and $\beta_i = 10^8$, to penalize the controller very heavily for writing data to the hard disk. The decision variables μ_i and ω_i are quantized in intervals of 0.1. The controller sampling time T is set to 80 seconds in the implementation.

The *ADSS Element Manager* supplies the controller with internal state of the ADSS and SS services, including the observed buffer size on each node, n_i the simulation-data generation rate and the network bandwidth. The effective network bandwidth of the link between NERSC and PPPL is measured using Iperf [67], which reports the TCP bandwidth available, delay jitter and datagram loss.

The element manager also stores a set of rules, which are triggered based on controller decisions and enforce adaptations within the DTS/BMS. For example, the controller decides the amount of data to be sent over the network or to local storage and the element manager decides the corresponding buffer management scheme to be used within the BMS to achieve this. The element manager also adapts the DTS service to send data to local/low latency storage, example, NERSC/ORNL, when the network is congested.

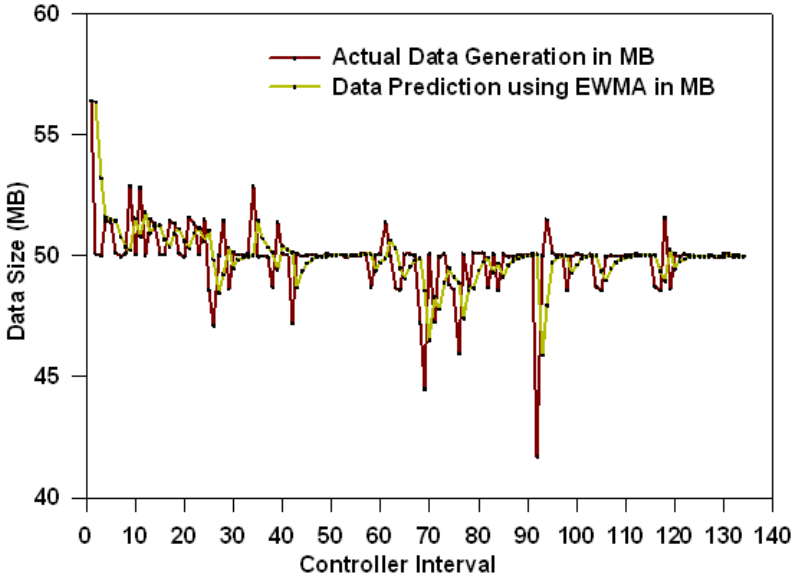

Figure 6.7: Actual and Predicted Data Generation Rates for the GTC simulation

6.2.3 Performance Evaluation

The setup for experiments presented in this section consisted of the GTC fusion simulation running on 32 to 256 processors at NERSC and streaming data for analysis to PPPL. A 155 Mbps ESNET connection between PPPL and NERSC was used. A single controller was used and the controller and managers were implemented using threading. Up to four simulation processors were used for data streaming.

Predicting data generation rates:

Figure 6.7 compares the actual amount of data generated by the simulation against the corresponding estimation. The simulation ran for three hours at NERSC on 64 processors and used four data streaming processors. The incoming data rate into each transfer processor was estimated with good accuracy by a EWMA filter as follows: $\hat{\lambda}_i(k) = \gamma.\lambda_i(k) + (1-\gamma).\hat{\lambda}_i(k-1)$, where $\gamma = 0.5$ is the smoothing factor. It follows from the plot that the EWMA can accurately predict the data generation for GTC simulations.

Controller behaviour for long-running simulations:

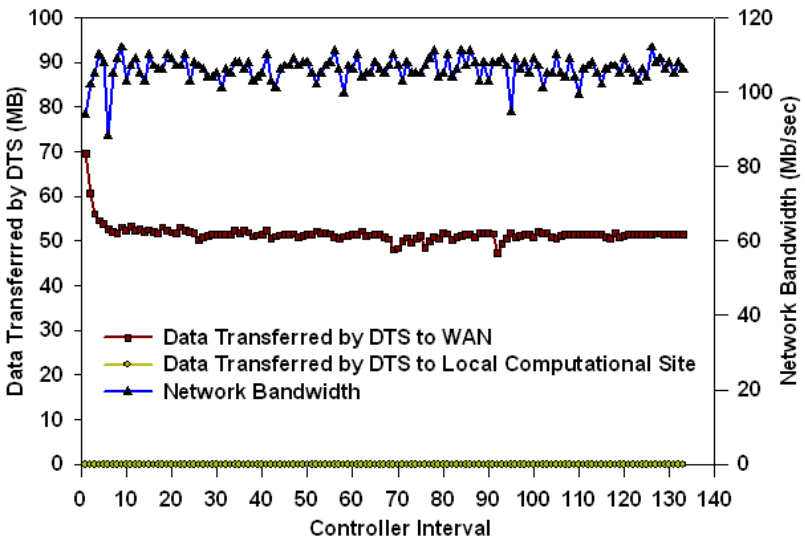

Figure 6.8: Controller and DTS operation for the GTC simulation

Figure 6.8 plots a representative snapshot of the streaming behaviour for a long-running GTC simulation. During the shown period, DTS always transfers data to remote storage and no data is transferred to local storage, as the effective network bandwidth remains steady and no congestions are detected. This plot illustrates the stable operation of the controller.

DTS adaptations based on control strategies:

To observe adaptation in the DTS, we congested the network between NERSC and PPPL between controller intervals 9 and 19 (recall that each controller interval is 80 sec), as shown in Figure 6.9. During intervals (1, 9), we observe no congestion in the network and data is transferred by DTS over the network to PPPL. During the intervals of network congestion (9, 18), the controller observes the environment and state variables and advices the element manager to adapt the DTS behaviour accordingly, causing some data to be transferred to a local storage/hard disk in addition to sending data to the remote location. This prevents data loss due to buffer overflows. It is observed from Figure 6.9 that this adaptation is triggered multiple times until the network is no longer congested at around the 19^{th} controller interval. The data sent to the local storage falls to zero at this point.

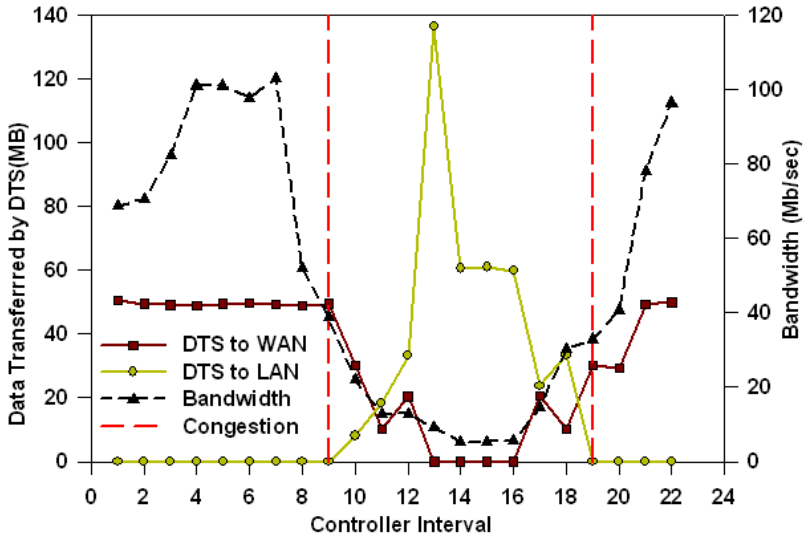

Figure 6.9: DTS Adaptation due to Network Congestion

Adaptations in the BMS:

This scenario demonstrates the adaptation of the BMS service. A uniform BMS scheme is triggered in cases when data generation is constant and in cases when the congestion increases an aggregate buffer management is triggered. The triggering of the appropriate buffering scheme in the BMS is prescribed by the controller to overcome network congestion. Figure 6.10 shows the corresponding adaptations. During intervals (0, 7), the uniform blocking scheme is used and during (7, 16), the aggregate blocking scheme used to compensate for network congestion.

Comparison of Rule-based and Control-based Adaptation in the ADSS:

This evaluation illustrates how the percentage buffer vacancy (i.e., the empty space in the buffer) varies over time for two scenarios; one in which only rules are used for buffer management and the other in which rules are used in combination with controller inputs. Figure 6.11 plots the %buffer vacancy for the first case. In this case, management was purely reactive and based on heuristics (rule based). The element manager was not aware of the current and future data generation rate and the network bandwidth. The average buffer vacancy in this case was around 16%, i.e., in most cases 84% of the buffer was full.

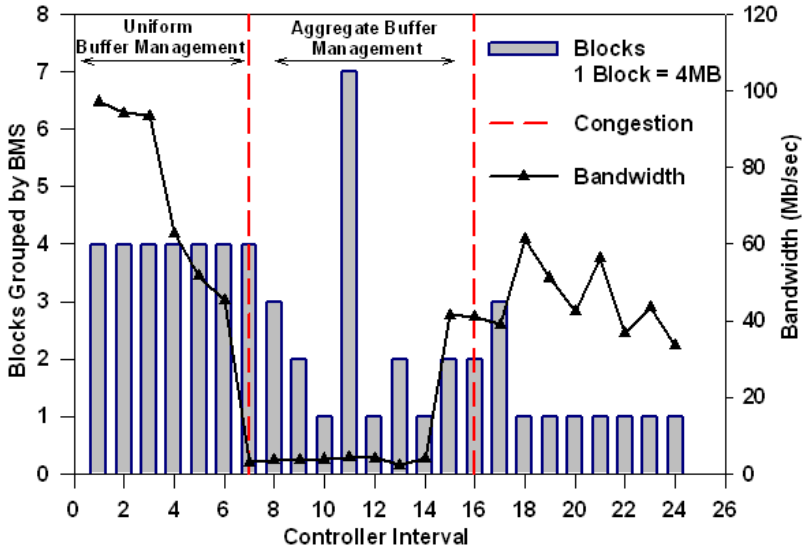

Figure 6.10: BMS Adaptations due to Varying Network Conditions

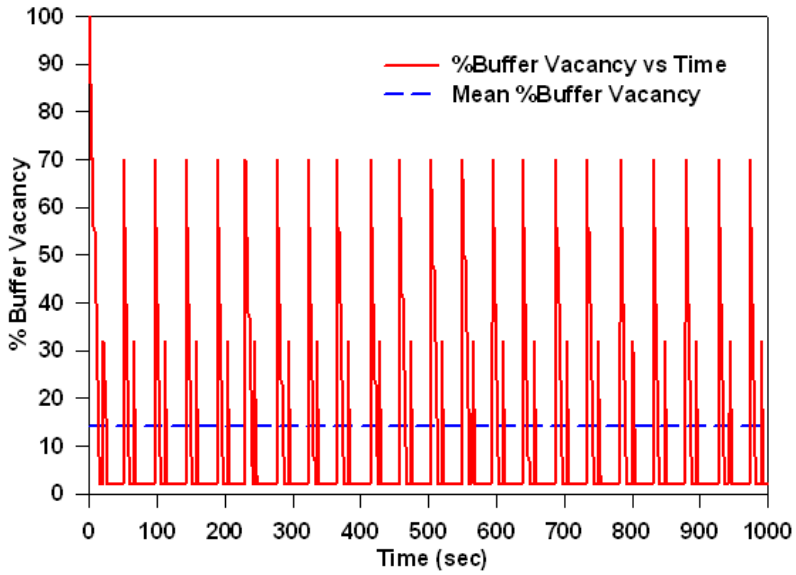

Figure 6.11: %Buffer Vacancy using Heuristically based Rules

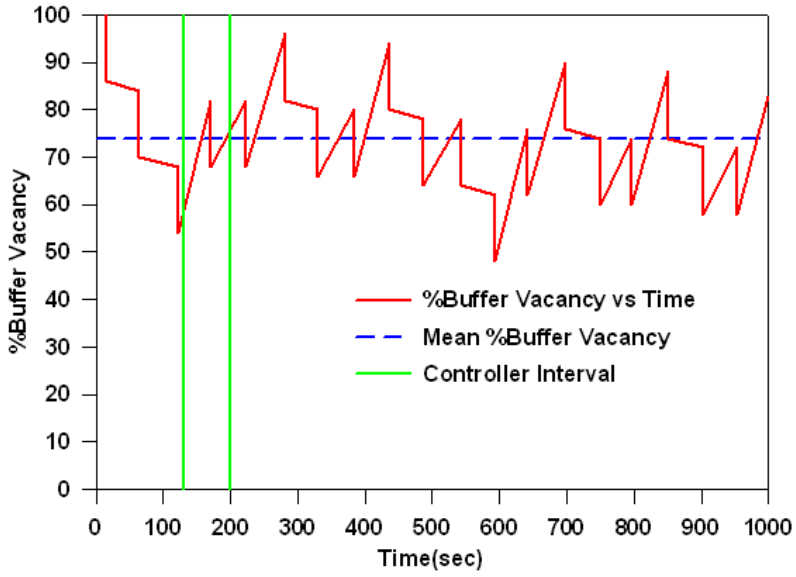

Figure 6.12: %Buffer Vacancy using Control-based Self-Management

Such a high occupancy leads to a slow down of the simulation [15] and also results in increased loss of data due to buffer overflows. Figure 6.12 plots the corresponding %buffer vacancy when the model-based controller was used in conjunction with rule-based management. The mean buffer vacancy in this case is around 75%. Higher buffer vacancy leads to reduced overheads and data loss.

Overhead of the Self-Managing Data Streaming:

Overheads on the simulation due the self-managing data streaming service are primarily due to two factors. The first are the activities of the controller during a controller interval. This includes the controller decision time, the cost of adaptations triggered by rule executions and the operation of BMS and DTS. The second is the cost of the data streaming itself. These overheads are presented below.

Overheads due to controller activities: For a controller interval of 80 seconds, the average controller decision-time was ≈2.1 sec (2.5%) at the start of the controller operation. This reduced to ≈0.12 sec (0.15%) as the simulation progressed due to local search methods used. The network measurement cost was 18.8 sec (23.5%). The operating cost of the BMS and DTS was 0.2 sec (0.25%) and 18.8 sec (23.5%)

respectively. Rule execution for triggering adaptations required less than 0.01 sec. The controller was idle for the rest of the control interval. Note that the controller was implemented as a separate thread (using *pthread* [66]) and its execution overlapped with the simulation.

Overhead of data streaming: A key requirement of the self managing data streaming was that its overhead on the simulation be less than 10% of the simulation execution time. %Overhead of the data streaming is defined as: $(\hat{T}_s - T_s)/T_s$, where $\hat{T}_s$ and T_s denote the simulation execution time with and without data streaming respectively. The %Overhead of data streaming on the GTC simulation was less than 9% for 16-64 processors and reduced to about 5% for 128-256 processors. The reduction was due to the fact that as the number of simulation processors increased, the data generated per processors decreased.

6.3 Addressing Scalability Using Hierarchical Control

In a distributed application consisting of multiple interacting elements, a centralized scheme for enforcing self-managing behaviours is not scalable - the number of control options to be explored is simply too large. However, the dimensionality of the overall optimization problem is drastically reduced, if it can be decomposed into simpler sub-problems, where each is solved independently. Higher-level control can be used to enable coordinated adaptations across these sub domains, as discussed below. To solve performance management problems of interest tractably in a distributed setting, service managers in Accord can be dynamically composed in hierarchical fashion, as shown in Figure 6.13, where interactions between element controllers are managed by higher-level ones. Decisions made by high-level controllers are aimed at satisfying overall QoS goals and act as additional operating constraints on lower-level elements. Each element optimizes its behaviour using its local controller, while satisfying these constraints. The Accord runtime framework ensures coordinated and

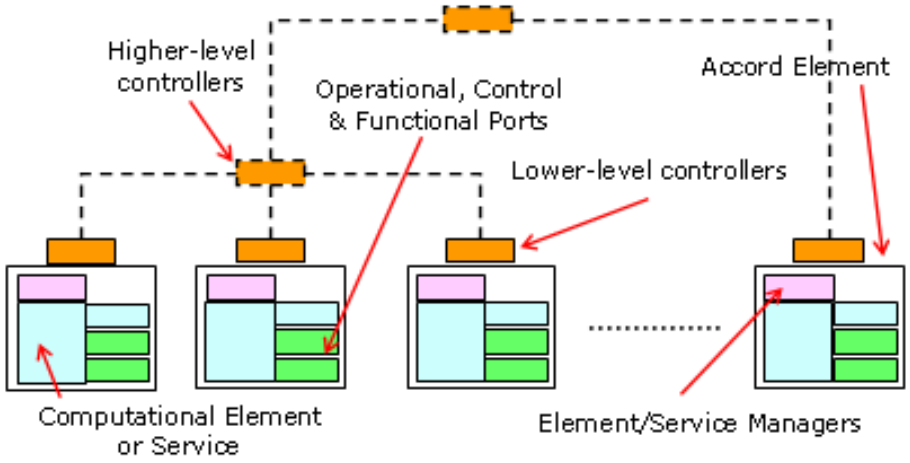

Figure 6.13: Constructing a Hierarchy of Controllers in the Accord Programming Framework

consistent adaptations across multiple service (element) managers. The overall operation is as follows. At runtime, each element or service manager independently collects element and context state information using sensors exposed by the individual elements and the environment. The managers then report this information to associated controllers, which then computes control actions and informs the service manager of desired adaptation behaviours. Service managers then execute these adaptation behaviours using actuators exposed by the environment and elements. If these local adaptations do not achieve the desired objectives, service managers collectively invoke higher-level controllers, which results in coordination among multiple interacting managers to change the element state and their interactions. Composition managers coordinate adaptations across service managers as described above.

6.3.1 Hierarchical Controller Design for Data Streaming

Recall that when control inputs must be chosen from a set of discrete values, the optimization problem described in Section 6.2.1 will show an exponential increase in worst-case complexity with an increasing number of control options and longer prediction horizons. We can, however, substantially reduce the dimensionality of

the optimization problem via *hierarchical control decomposition*. Exhaustive and bounded search strategies are then used at different levels of the hierarchy to solve the corresponding optimization problems with low run-time overhead. As an example of how to apply hierarchical control to the data streaming problem, consider the multi-level structure shown in Figure 6.14. Here, we have a larger system compared to the one described in Section 6.2.1 - 256 processors generate simulation data while 16 data-transfer nodes (instead of 4) collect this data and stream it over the network link to PPPL. As before, the QoS goals are to prevent any loss of simulation data and maximize the utility of the transferred data. First, the data-transfer nodes are logically partitioned, for the purposes of scalable control, into four modules M_1, M_2, M_3 and M_4 where each module M_i itself comprises four nodes. The data-generation or flow rate from the simulation cluster into each M_i at time k is denoted by $F_i(k)$. This flow can be further split into sub-flows $F_{i1}(k)$, $F_{i2}(k)$, $F_{i3}(k)$ and $F_{i4}(k)$, incoming into each node within module M_i.

Figure 6.14 shows $L1$ and $L0$ controllers within a two level hierarchy working together to achieve the desired QoS goals with the following responsibilities. The $L1$ controller must decide the fraction of the available network bandwidth to distribute to the various modules. Therefore, given the incoming flow-rates into the various modules, the effective network bandwidth $B(k)$ and the current state of each module in terms of the average buffer size of the sending processors, the L1 controller must decide the vector γ_i, i.e., the fraction of the network bandwidth $\gamma_i.B(k)$ to allocate to each M_i. The $L0$ controller within M_i solves the problem, originally formulated in Section 6.2. It decides the following variables for each node n_j in the module: the fractions μ_{ij} and ω_{ij} of the incoming flow rate $F_{ij}(k)$ to send over the network link and to the local/nearby storage, respectively. It is important to note that the $L0$ controller within a module operates under the dynamic constraints imposed by the $L1$ controller, in terms of the bandwidth $\gamma_i.B(k)$ that the $L0$ controller must distribute among its sending processors.

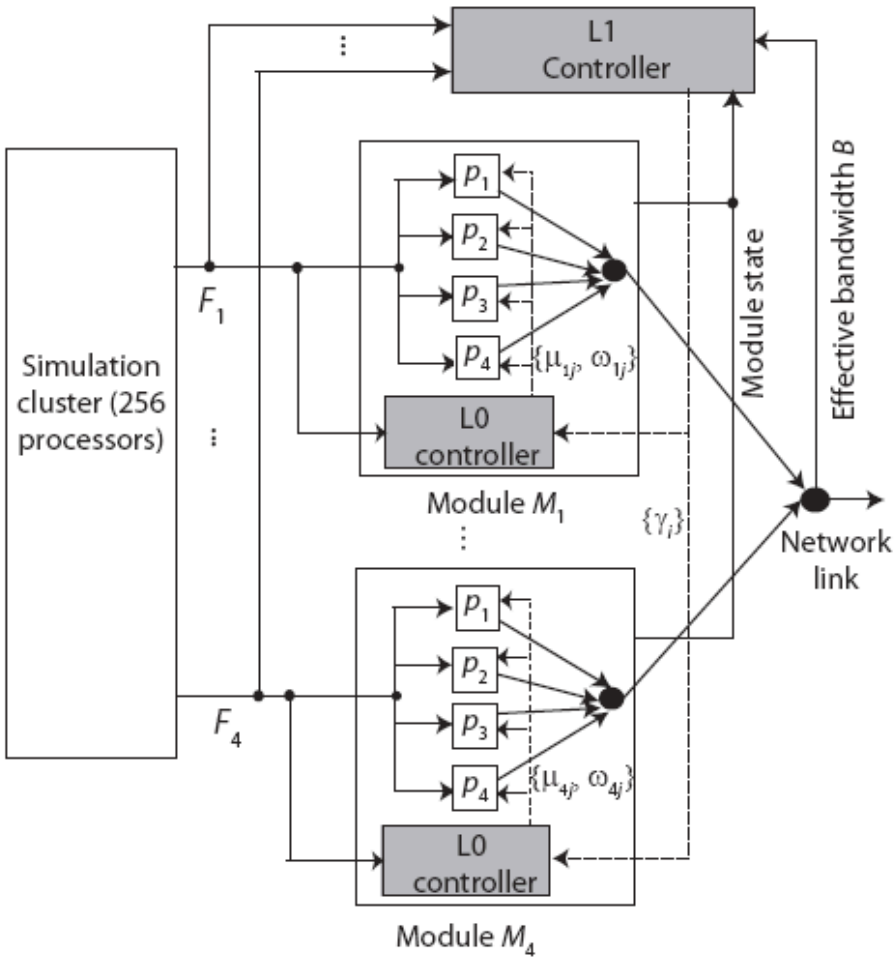

Figure 6.14: Hierarchical Controller Formulation for Data Streaming

The hierarchical structure in Figure 6.14 reduces the dimensionality of the original control problem substantially. Where a centralized solution must decide the variables μ and ω for each of the 16 sending processors, in our method, the $L1$ controller only decides a single-dimensional variable γ for each of the four modules. Similarly, the $L0$ controller decides control variables only for those processors within its module - far fewer compared to the total number of sending processors in the system.

To realize the hierarchical structure in Figure 6.14, each $L1$ controller must know the approximate behavior of the components comprising the $L0$ level. For example, to solve the combinatorial optimization problem of determining γ_i, the fraction of the available network bandwidth to allocate to the modules, the $L1$ controller must be

able to quickly approximate the behavior of each module. More specifically, given the observed state of each M_i and the estimated environment parameters in terms of the effective network bandwidth and flow rates, the $L1$ controller must obtain the cost incurred by module M_i for various choices of γ_i. Note, however, that M_i's behavior includes complex and non-linear interaction between its $L0$ controller and the corresponding sending processors and the resulting dynamics cannot be easily captured via explicit mathematical equations. A detailed model for each M_i will also increase the $L1$ controller's overhead substantially, defeating our goal of scalable hierarchical control.

We use *simulation-based learning* techniques [13] to generate a look-up table that quickly approximates M_i's behavior. Here, M_i's behavior is learned by simulating the module with a large number of training inputs from the (quantized) domains of F_i, B and γ_i. Once such an approximation is obtained off-line, it can be used by the $L1$ controller to generate decisions fast enough for use in real time.

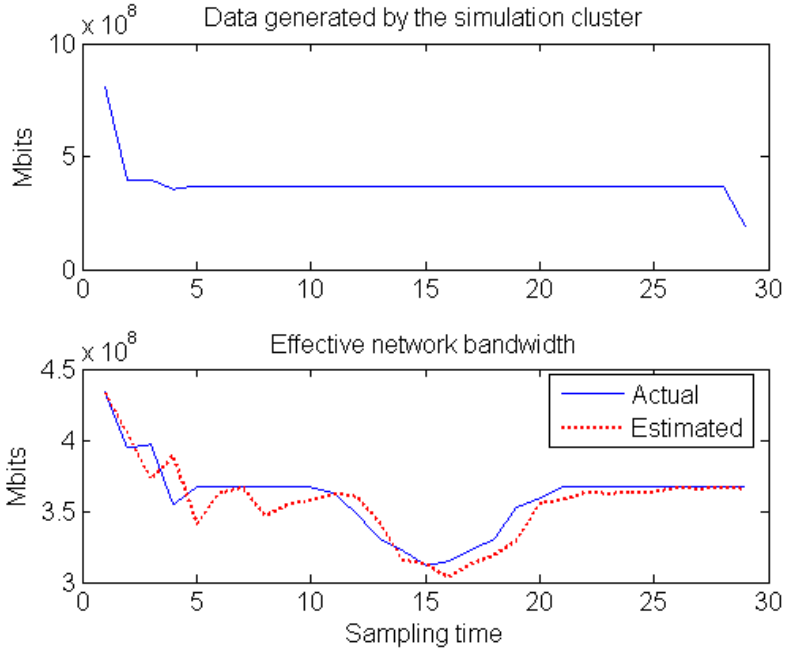

Figure 6.15: GTC Workload Trace and Effective Bandwidth between NERSC and PPPL

6.3.2 Simulation Results for Hierarchical Data Streaming

Figure 6.15 shows a workload trace representing the data generated by a simulation cluster comprising 256 processors and the effective network bandwidth available for data transfer between NERSC and PPPL. Both traces are plotted with a time granularity of 120 seconds. Note that though the data generation rate holds steady, the effective network bandwidth shows time-of-day variations. For example, the network is somewhat congested during the time steps 12 to 18. Both the data generation rate and the effective bandwidth can be estimated effectively using an ARIMA (AutoRegressive Integrated Moving Average) filter with properly tuned smoothing parameters.

Figure 6.16 summarizes the performance of the control hierarchy when both the $L0$ and $L1$ controllers use a single step look-ahead LLC scheme. We assume a total of 16 data transfer nodes, arranged in four modules comprising four nodes each. The sampling times for the $L0$ and $L1$ controllers are both set to 120 seconds. The maximum buffer size on each node was $q_{max} = 3.10^7$ bits (≈29MB) and the desired queue size at the end of the prediction horizon was set to $q^* = 0$. The decision variable $0 \leq \gamma_i \leq 1$ supplied by the L1 controller to each M_i was quantized in intervals of 0.1. Figure 6.16 shows the data, in terms of Mbits, streamed by the $L0$ controller within each module over the network link and hard disk. It is clear that during periods of network congestion, between 12 and 18, the $L0$ controllers within modules M_1 and M_3 write a fraction of the incoming data to hard disk to prevent data loss.

6.4 Conclusions

The chapter presented the design and implementation of a self-managing data streaming service that enables efficient data transport to support emerging Grid-based scientific workflows. The presented design combines rule-based heuristic adaptations with more formal model-based online control strategies to provide a self-managing service framework that is robust and flexible and can address the dynamism in application

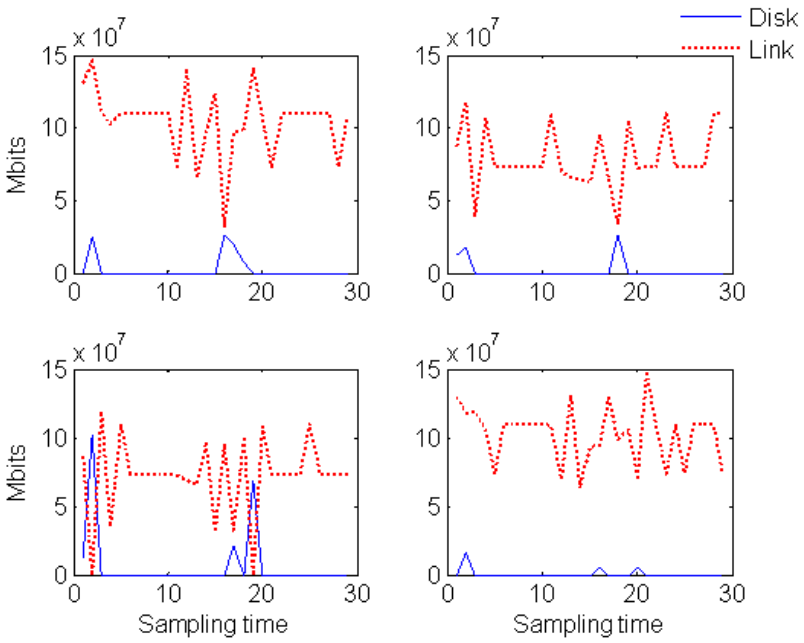

Figure 6.16: Operation of the L0 and L1 controllers

requirements and system state. A fusion simulation workflow was used to evaluate the data-streaming service and its self-managing behaviours. The results demonstrate the ability of the service to meet Grid-based data-streaming requirements, as well as its efficiency and performance. A hierarchical control architecture was also presented to address scalability issues for large systems. Simulations were used to demonstrate the feasibility and effectiveness of the scheme.

Chapter 7

Experiments with In-Transit Processing for Data Intensive Grid Workflows

The Grid cyberinfrastructure is rapidly enabling new data intensive scientific and engineering application workflows, which are based on seamless interactions and coupling between geographically distributed application components. For example, a typical fusion simulation consists of coupled codes running simultaneously on separate HPC resources at supercomputing centers, and interacting at runtime with additional services for interactive data monitoring, online data analysis and visualization, data archiving, and collaboration. A key requirement of these applications is the support for asynchronous, high-throughput low-latency robust data streaming between the interacting components. The fusion codes, for instance, require continuous data streaming from the HPC machine to ancillary data analysis and storage machines. Moreover, these data-streaming services must deal with high data volumes and data rates, have minimal impact on the execution of the simulations, deal with natural mismatches in the ways data is represented in different program components and on different machines, be able to "outsource" data manipulation and transformation operations to less expensive commodity resources in the data path while satisfying stringent application/user space and time constraints, and guarantee that no data is lost. Satisfying these requirements presents many challenges, especially in large-scale and highly dynamic environments with shared computing and communication resources, resource heterogeneity in terms of capability, capacity and costs, and where application behaviour, needs, and performance are highly variable.

The overall goal of this research to develop a data streaming and in-transit data

manipulations service that provides the mechanisms as well as the management strategies for data intensive scientific and engineering workflows to addresses the requirements outlined above. In our previous work we addressed efficient and robust wide-area data streaming, and developed autonomic management strategies based on online control [16]. The developed service minimizes overheads on the simulations, provided proactive model-based Quality of Service (QoS) control at the data source, and avoids loss of data.

In this chapter, we address in-transit data manipulation and transformation using resources in the data path between the source and the destination. The specific objectives of this chapter are (1) to experiment with reactive management strategies for in-transit data manipulation, and (2) to investigate the coupling of these strategies with the application level self-managing data streaming service developed in our previous work, to create a cooperative management framework for wide-area data-streaming and in-transit data manipulation for data-intensive scientific and engineering workflows. This research is driven by the requirements for the Department of Energy (DOE) Scientific Discovery through Advanced Computation Solicitation (SciDAC), Center for Plasma Edge Simulation (CPES) Project [48] and the Grid-based coupled fusion simulations that are used in the experiments presented in this chapter.

The rest of this chapter is organized as follows. Section 7.1 describes the driving Grid-based fusion simulation project and highlights its data streaming and in-transit data processing challenges and requirements. Section 7.2 presents the overall architecture of the proposed data streaming service and the cooperative management framework, and summarizes our previous work on autonomic application level data streaming using model based online control. Section 7.3 describes the in-transit data manipulation framework and presents experimental evaluations of various strategies for managing the in-transit operation and cooperative end-to-end management. Section 7.4 concludes the chapter and presents future work.

7.1 The Fusion Simulation Project and its Data Streaming Requirements

7.1.1 Fusion Simulation Workflow

The overarching goal of the DOE SciDAC CPES fusion simulation project [48] is to develop a new integrated Grid-based predictive plasma edge simulation capability to support next-generation burning plasma experiments, such as the International Thermonuclear Experimental Reactor (ITER). Effective online management and transfer of the simulation data is a critical part for this project and is essential to the scientific discovery process. It consists of coupled simulation codes, i.e., the edge turbulence particle-in-cell (PIC) code XGC coupled with Nimrod and the microscopic MHD code (M3D), which run simultaneously on thousands of processors on separate HPC resources at possibly distributed supercomputing centers. The data produced by these simulations must be streamed at runtime, to remote sites, for online simulation monitoring and control, simulation coupling, data analysis and visualization, online validation, and archiving. Furthermore, the data may have to be processed enroute to the destination as the data may have to transformed to match the coordinate system, representation, format, distribution and mapping, etc., of the destination node. Similarly, features of interest may need to be extracted and processed enroute to visualization or monitoring applications.

7.1.2 Data Streaming and In-Transit Processing Requirements

The fundamental requirement for a wide area data streaming and in-transit data processing service is to efficiently and robustly stream data from live simulations to remote services while satisfying the following constraints: (1) Enable high-throughput, low-latency data transfer to support near real-time access to the data. (2) Minimizing overheads on the executing simulation. The simulation executes in batch for days and

we would like the overhead of the streaming on the simulation to be less than 10% of the simulation execution time. (3) Adapting to network conditions to maintain desired QoS. The network is a shared resource and the usage patterns typically vary constantly. (4) Handle network failures while eliminating loss of data. Network failures usually lead to buffer overflows, and data has to be written to local disks to avoid loss. This increases the overhead in the simulation. Further, the data is no longer available for remote analysis. (5) Effectively schedule and manage in-transit processing while satisfying the above requirements - this is particularly challenging due to the limited capabilities and resources and the dynamic capacities of the typically shared processing nodes.

7.2 A Self-Managing Service for Data Streaming and In-Transit Processing

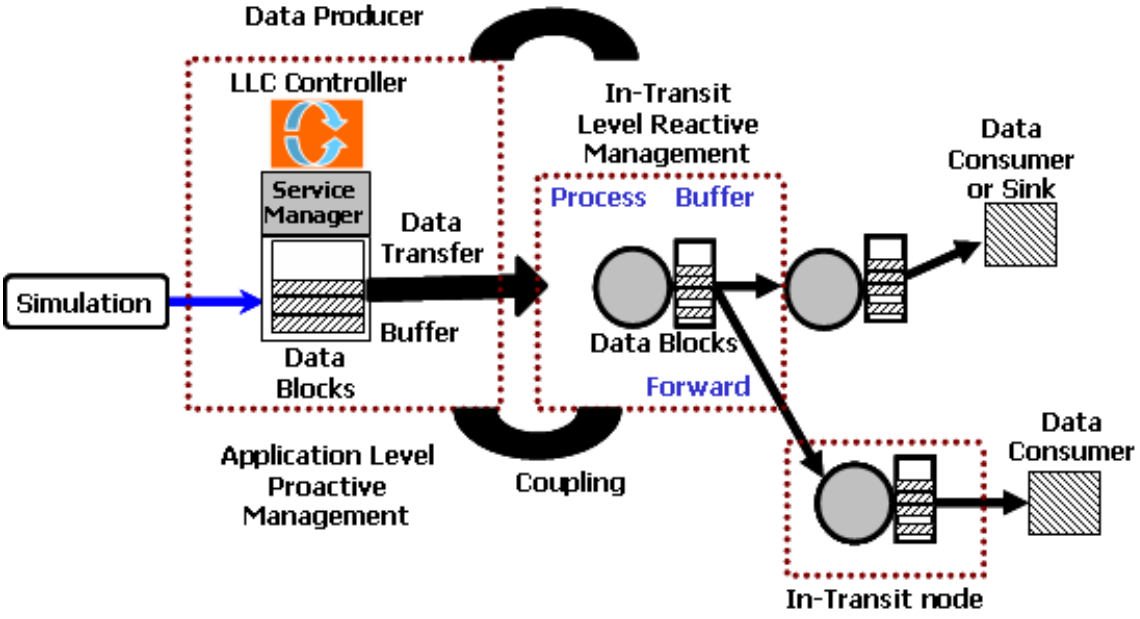

Figure 7.1: Conceptual Overview of the Self-managing Data Streaming and In-Transit Processing Service

A conceptual overview of the self-managing data streaming and in-transit processing service for Grid-based data intensive scientific workflows is presented in Figure 7.1. It consists of two key components: The first is an application level data streaming service, which provides adaptive buffer management mechanisms and proactive QoS management strategies based on online control and user-defined polices, at application end-points. The second component provides scheduling mechanisms and adaptive runtime management strategies for in-transit data manipulation and transformation. These two components work cooperatively to address the overall application constraints and QoS requirements outlined in Section 7.1.2. The first component has been addressed in our previous work [16] and is briefly summarized below. This chapter focuses on the second component and experiments with different in network processing strategies as well as their couplings with the application level mechanisms.

7.2.1 Application Level Data Streaming

The application level self-managing data streaming service combines model-based limited look-ahead controllers (LLC) and rule-based autonomic managers with adaptive multi-threaded buffer management and data transport mechanisms at the application endpoints. It is constructed using the Accord-WS infrastructure for self-managing Grid services [53] and supports high throughput, low latency, robust application level data streaming in wide-area Grid environments as demonstrated in [16]. The autonomic data streaming service is illustrated in Figure 7.2 and consists of a service manager and an LLC controller. The service manager monitors the state of the service and its execution context, collects and reports runtime information, and enforces the adaptation actions determined by its controller. Augmenting the element manager with an LLC controller allows human defined adaptation polices, which may be error-prone and incomplete, with mathematically sound models and optimization techniques for more robust self-management. Specifically, the controller decides when and how to adapt the application behavior and the service managers focus on

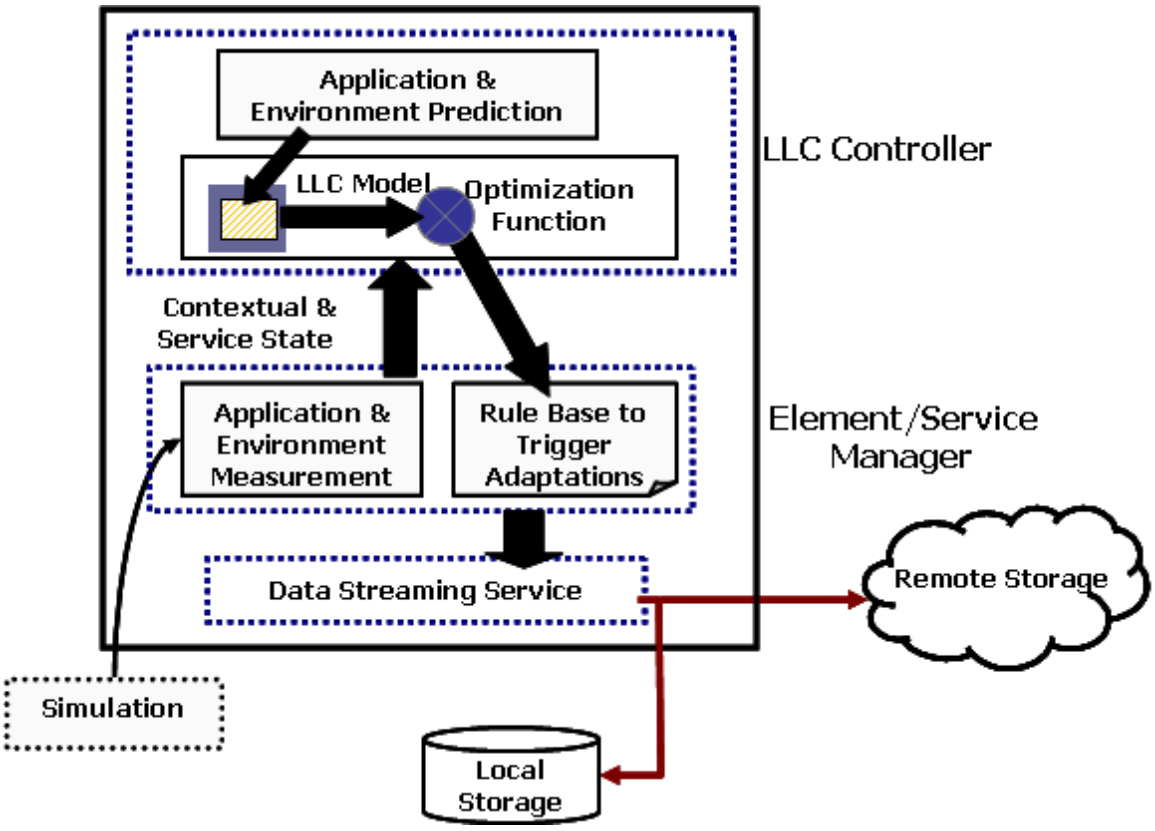

Figure 7.2: A Self-Managing Application Level Data Streaming Service

enforcing these adaptations in a consistent and efficient manner. The structure of the LLC-based online controller is shown in Figure 7.3. The figure shows the key operating parameters for the controller at simulation node n_i at time step k which are as follows. (1) State variable: The current average buffer size at n_i denoted as $q_i(k)$. (2) Environment variables: $\lambda_i(k)$ denotes the data generation rate into the buffer q_i and $B(k)$ the effective bandwidth of the network link from source to the sink. (3) Control or decision variables: Given the state and environment variables at time k, the controller decides $\omega_i(k)$ and $\mu_i(k)$, the data-transfer rate over the remote storage (Data Grid) and to the local storage respectively [16]. The objective of the controller denoted by q^* is to keep the %buffer occupancy $q_i(k)$ (%data blocks in the buffer) at zero. Note that $q_i(k)$ should be less than 100% so that the buffer does not overflow.

The self-managing service behaves as follows. The element manager supplies the

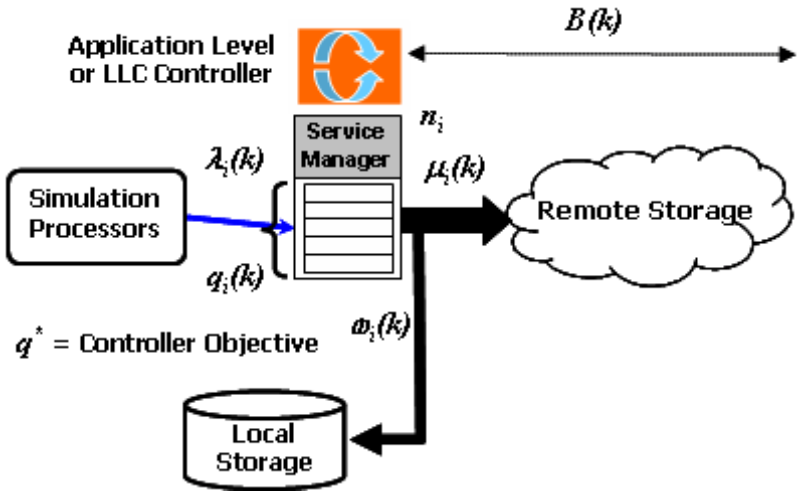

Figure 7.3: Design of the LLC controller for an Application Level Data Streaming Service

LLC controller (as shown in Figure 7.2) with information about the internal state of the application and the environment, which includes the observed buffer size, simulation-data generation rate, and the network bandwidth. When the controller detects congestion due to a decrease in parameter $B(k)$, it advises the service manager to increase $\omega_i(k)$ and decrease $\mu_i(k)$ to avoid loss of simulation data. The element manager contains the set of rules, which are invoked based on the controller's advice or decisions to adapt the service. For example, the controller decides the amount of data to be sent over the network or to local storage, and the service manager uses the controllers advise to select the corresponding buffer management scheme to be used within the data streaming service to achieve this. The element manager can also adapt the data streaming service to send data to local storage rather than streaming it to a remote site when the network is congested. Experimental evaluation of the application level data streaming service in a wide area Grid environment demonstrate its scalability, stability, its ability to effectively maintain application QoS and avoid data loss, as well as its low overheads on the simulation [16].

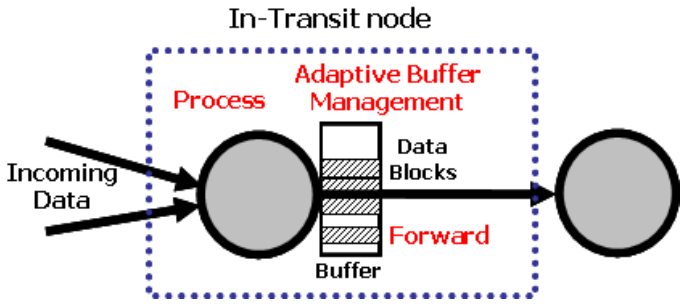

Figure 7.4: Architecture of an In-Transit Node

7.2.2 In-Transit Data Manipulations

The in-transit data manipulation framework consists of a dynamic overlay of available in-transit processing nodes (e.g., workstations or small to medium clusters) with heterogeneous capabilities and loads. Note that these nodes may be shared across multiple workflows. The conceptual architecture of a node is illustrated in Figure 7.4. Each node performs three steps, viz., processing, buffering and forwarding. The processing depends on the capacity and capability of the node and the amount of processing that is still required. The basic idea is that each node completes at least its share of the processing (which may be predetermined or dynamically computed) and can perform additional processing if the network is too congested for forwarding. The amount of processing completed is logged in the data block itself. The goal of the in-transit processing is to process as much data as possible before the data reaches the sink. A processing that is not completed in-transit will have to be performed at the sink. The current design of the framework assumes that each node can perform any of the required data manipulations functions. Each in-transit node maintains a buffer associated with each flow. The structure of this buffer is shown in Figure 7.5. The buffer has a fixed size and wraps around once it fills up. The data input rate at each in-transit node is the amount of data queued at the buffer per second and the buffer drainage rate is proportional to the network connectivity of the outgoing link. The buffering algorithm at the node is reactive in that it attempts to dynamically adjust to the buffer input and buffer drainage rates. It does this by aggregating the blocks

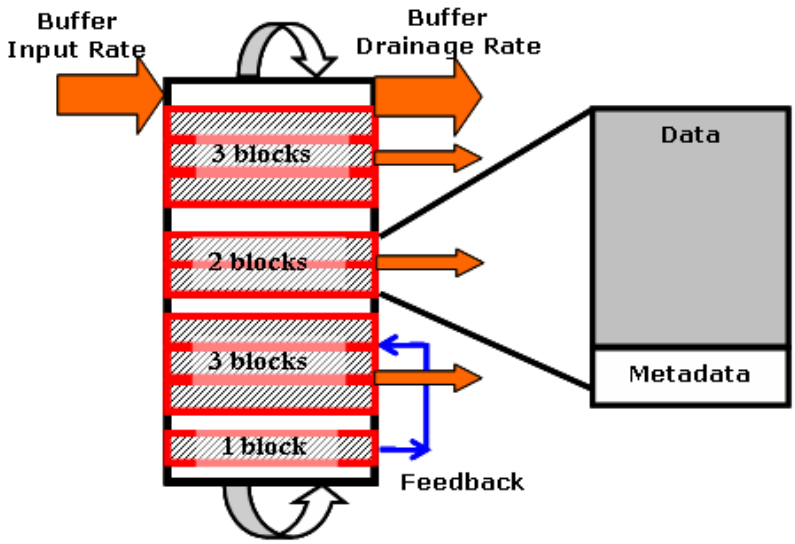

Figure 7.5: Adaptive Buffering at the In-Transit Node

of data that have accumulated since the start of the transfer and transfers this aggregated block of data in the next transmission. The size of the block transferred thus depends on the network connectivity and the transfer time of the previous transfer. Data transmission is multi-threaded and the number of transmission threads is controlled dynamically. Depending on the data input and drainage rates, the following situations can occur:

- Input rate exceeds drainage rate: In this situation the node attempts to maximize the data sent out by increasing the level of multi-threading at the transmission layer and improves throughput.

- Input rate is approximately equal to the drainage rate: In this situation new data accumulates in the buffer during each transfer. The first transfer will be the first data block queued, and the subsequent transfers will consist of blocks aggregated during the previous transfer. The buffer management scheme subsequently achieves equilibrium on the number of blocks transferred.

- Input rate is smaller than the drainage rate: In this situation, if the buffer manager encounters an empty buffer, it waits until more data is queued.

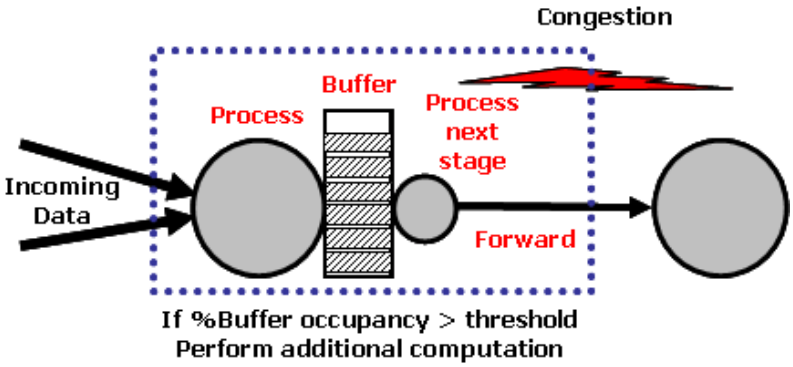

Figure 7.6: Adaptive Processing of Data at In-Transit Nodes in Response to Network Congestions

The operation of an in-transit node is a follows. Each incoming data block is first processed, then queued in the buffer and finally forwarded to the next stage. Thus, the time spent by a data block at each in-transit node is thus the sum of the processing time (t_p), buffering time (t_{buff}) and forwarding time (t_f). During congestion, t_{buff} can sharply increase in relation to t_p and t_f. Since congestion can cause buffer overflows and loss of data at the in-transit nodes. In this case, rather the node attempts to further process the data block. The heuristic used is based on %Buffer Occupancy, i.e. the %data blocks stored in the buffer - when a node's buffer occupancy exceeds a certain threshold; the node decides to perform additional computation on the data blocks. This is illustrated in Figure 7.6. The rationale is that subsequent in-transit nodes downstream or the sink will then have to perform a smaller processing, which will offset the increased latency due to congestion.

7.2.3 Cooperative Self-Management: Coupling Application Level and In-Transit Management

The application level and in-transit management can be coupled to achieve cooperative end-to-end self-management. Coupling is beneficial particularly in cases of congestion, which normally occur at one of the shared links in the data path between the sources and sink nodes.

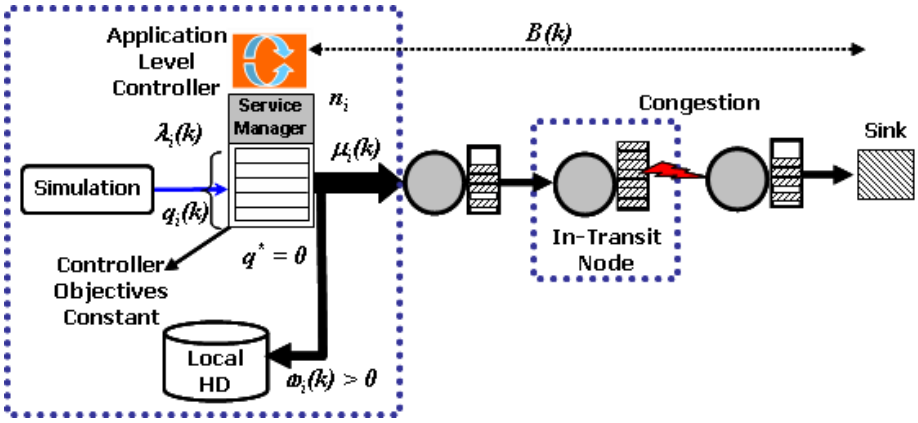

Figure 7.7: Application Level Management in Response to Network Congestions (without Coupling)

In the standalone case as illustrated in Figure 7.7, if application level management was used in isolation, the application level controller would detect the congestion by observing a decrease of parameter $B(k)$, and it would advise the service manager to increase $\omega_i(k)$ and decrease $\mu_i(k)$, i.e., to reduce the amount of data sent on the network and increase the amount of data written to the local storage thereby avoiding data loss. While this would eventually reduce the congestion in the data path, it would require that the data blocks written to the local storage be manually transferred to and processed at the sink.

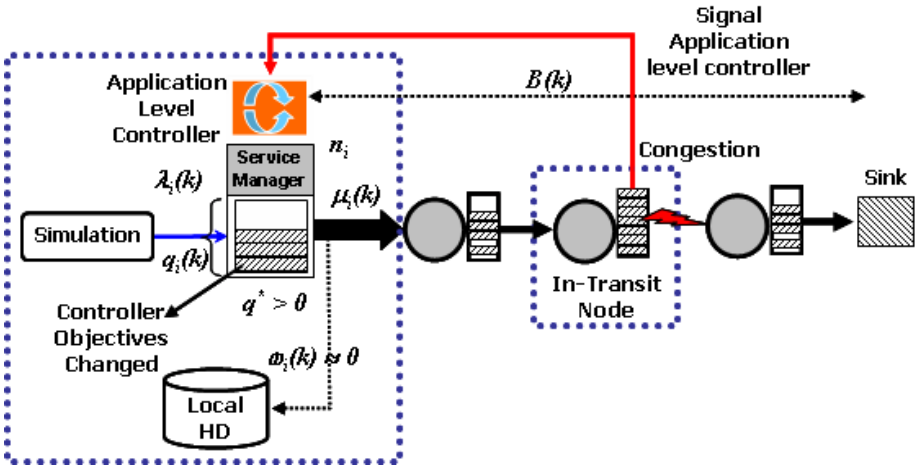

Figure 7.8: Cooperative End-to-End Management - In-Transit Node Signals Application Level Controller about Network Congestions (with Coupling)

However in the coupled scenario (see Figure 7.8), the in-transit node signals the controller at the source in response to local congestion that it detects by observing its buffer occupancy and sends it information about its current buffer size. This allows the application level controller to detect congestion more rapidly, rather than have to wait until the congestion propagates back to the source, and in response, it increases its $q_i(k)$ (or in turn q^*) to a value higher than zero so as to throttle items in its buffer till the congestion at the in-transit nodes is relieved. This, in turn, reduces the amount of data that is written to the local disk at the source.

7.3 Implementation and Experiments

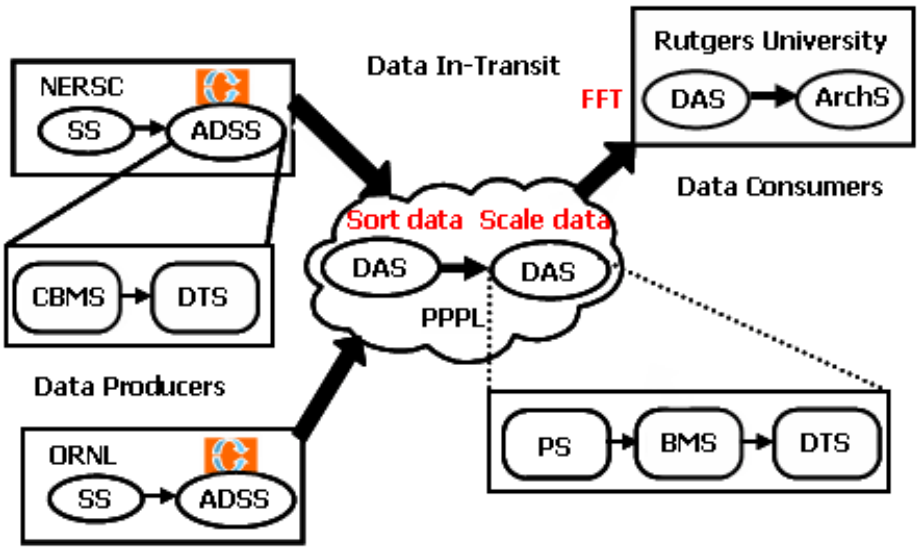

Figure 7.9: The Fusion Simulation Workflow used in the Experiments

This section presents experiments using the cooperative self-managing data streaming service as part of a fusion workflow. The overall application setup is shown in Figure 7.9. It consists of the Simulation Service (SS), i.e., the GTC fusion simulation, which runs at NERSC (CA) and ORNL (TN), and streams data for analysis to PPPL (NJ) and final data archiving at Rutgers University (NJ). The simulation service (SS) executes on 32 to 256 processors on "Seaborg", an IBM SP machine at NERSC, and on 256 processors on "RAM", an SGI Altix machine. The Autonomic (self-managing)

Data Streaming Service (ADSS) is co-located with the SS at NERSC and ORNL. The in-transit processing is performed by the Data Analysis Service (DAS) located at the in-transit nodes at PPPL and Rutgers. Data Archiving Service (ArchS) is also located at Rutgers which is referred to as the sink or data consumer. Three in-transit nodes were used in these experiments. These included 32 AMD Athlon MP 2100+ processors ("gridn" cluster), 4 dual-core AMD Opteron processors ("portalx" cluster) both located at PPPL and a 64 processor Intel Pentium (1.70GHz) Beowulf cluster ("Frea") located at Rutgers. Note that there is a 155 Mbps (peak) ESNET [50] connection between PPPL and NERSC and a 100 Mbps network connection between PPPL and Rutgers.

The ADSS service consists of a Controller based Buffer Management Service (CBMS), which contains an LLC online controller, and a Data Transfer Service (DTS). The controller interval for the CBMS was set to 80 seconds based on the data generation rates at the simulation end [16]. DTS uses a generic high performance transfer library for transferring data from simulation machines and is based on Logistical Networking (LN) [73].

The Data Analysis Service (DAS) operating at PPPL and Rutgers consists of the Processing Service (PS), Reactive Buffer Management Service (BMS) and Data Transfer Service (DTS). DAS consumes data blocks streamed from the simulation or adjacent DAS services, and after applying the right PS it forwards them to the following DAS. Three in-transit processing functions were used in these experiments, viz., sorting, scaling and FFT, each of which could be run on any of the in-transit nodes. The experiments conducted are presented below.

7.3.1 Normal Operation of DAS without Congestion

This experiment evaluates the behavior of DAS at the in-transit nodes during normal operation, i.e., when there is no congestion. Figure 7.10 plots the average relative times spent per data block on each of the three component services, i.e., processing

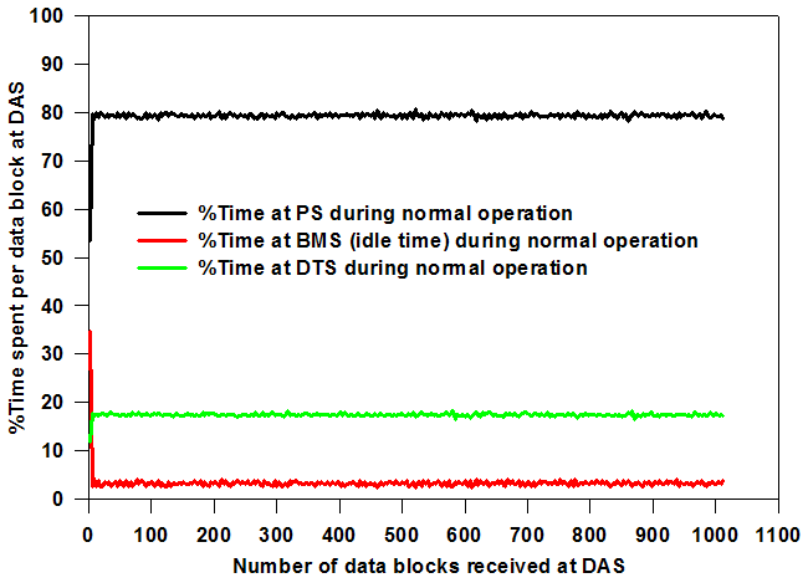

Figure 7.10: Breakup of the %Time Spent at the Each of the Services Comprising the DAS per Data Block

(PS), buffering (BMS) and forwarding (DTS). As seen from the figure, processing time (i.e., PS) is 80% on average, buffering time (i.e., BMS) is 3.2% on average, and forwarding time (i.e., DTS) is 17.8% on average. Buffering time mainly denotes the idle time for the data block in the DAS. Note that during the initial phases of the experiment, it is observed that the BMS time is significantly higher because of initial buffer warm up. This experiment provides the baseline for the experiments presented below.

7.3.2 Operation of the DAS during Congestion but without Adaptation

In this experiment, congestion was introduced between PPPL and Rutgers using the Trickle library [34], and the experiments conducted above were repeated. Figure 7.11 once again plots the average relative times spent per data block on each of the three component services, i.e., processing (PS), buffering (BMS) and forwarding (DTS). The plots show that during congestion, the forwarding time increases significantly

when compared to the normal operation case (i.e., Figure 7.10) and accounts for 41.64% of the total time. The buffering time (i.e., BMS) also increases as expected. Since there is no adaptation, the processing component (i.e., PS) remains the same but only accounts for 33% of the total time in this case.

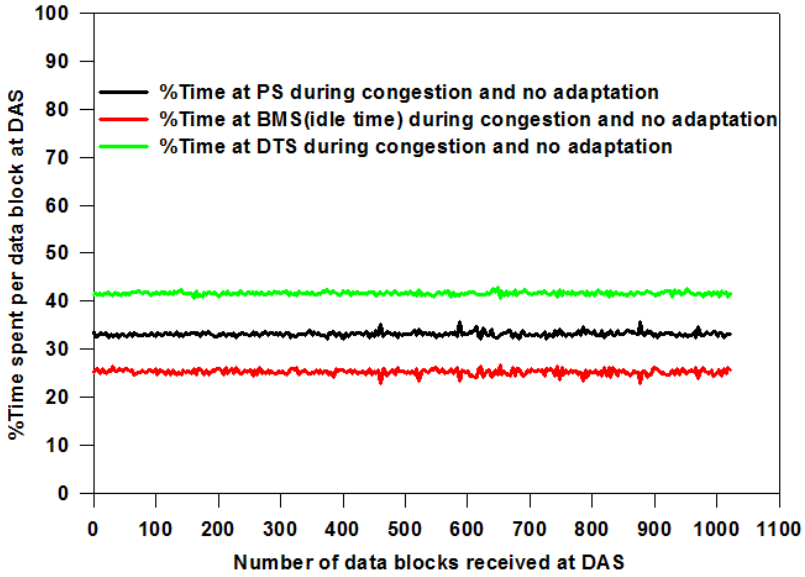

Figure 7.11: Breakup of %Time Spent at the Each of the Services Comprising the DAS per Data Block During Congestion and No Adaptation

7.3.3 Operation of DAS during Congestion with Adaptation

This experiment modifies the experiment above to introduce adaptation at the in-transit nodes, i.e., the DAS service adaptively processes data in its buffers when it observes the %buffer occupancy is above 60%. As seen in Figure 7.13, the buffering (BMS) time decreases and the processing (PS) time increases correspondingly as expected. The adaptation does not effect the forwarding (DTS) time. The effects of the adaptation can be seen in Figure 7.12. In this figure, the buffering (BMS) time (thick lines in the graph) reduces from an average of 1.2 seconds in the case without adaptation to an average of 0.06 seconds with adaptation. The overall time per data block in the DAS is slightly reduced from 4.83 seconds to 4.53 seconds as data blocks would have to be written to high latency local storage without adaptation.

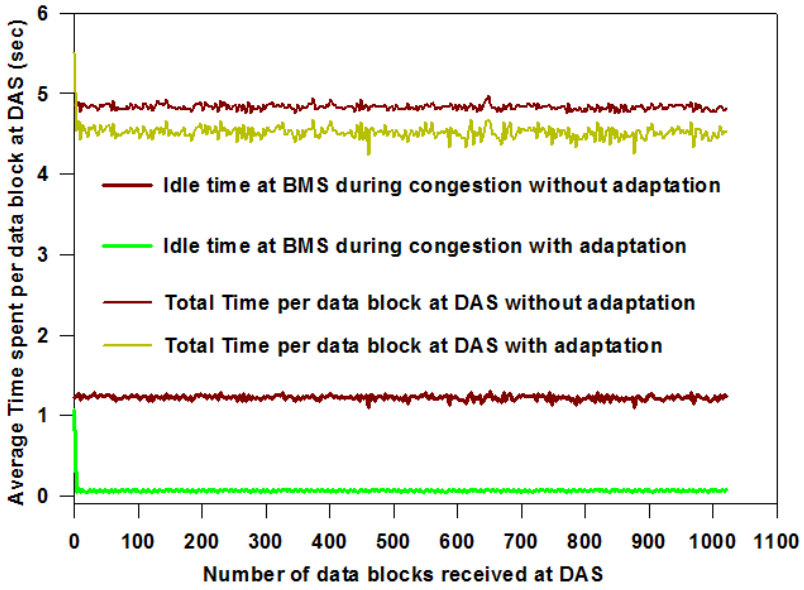

Figure 7.12: Effects of Adaptation on DAS During Congestion - Buffering or Idle Time Reduced Significantly

7.3.4 Operation of ADSS with and without Coupling

This experiment evaluates the end-to-end behavior of the application level ADSS service with and without cooperative management and coupling with the in-transit DAS service. The cumulative data transferred for different controller intervals for the two cases are plotted in Figure 7.14. Since congestion events sent by the in-transit nodes cause ADSS to buffer data blocks rather than having them written to local storage, the effective cumulative data transferred during congestion (i.e., controller intervals 9-20) drops. Figure 7.15 plots the average %buffer occupancy at an in-transit node (averaged over the three in-transit nodes used in the experiments) before, during and after congestion for this experiment. The average %buffer occupancy before the congestion is between 48.2% and 51%, which corresponds to normal operation (slight increase is due to the overheads of adaptation). During congestion, ADSS decides to throttle data blocks in response to congestion events from the in-transit nodes. This causes the average %buffer occupancy to decrease to about 60.8%. Without throttling and coupling, the average %buffer occupancy is significantly higher above

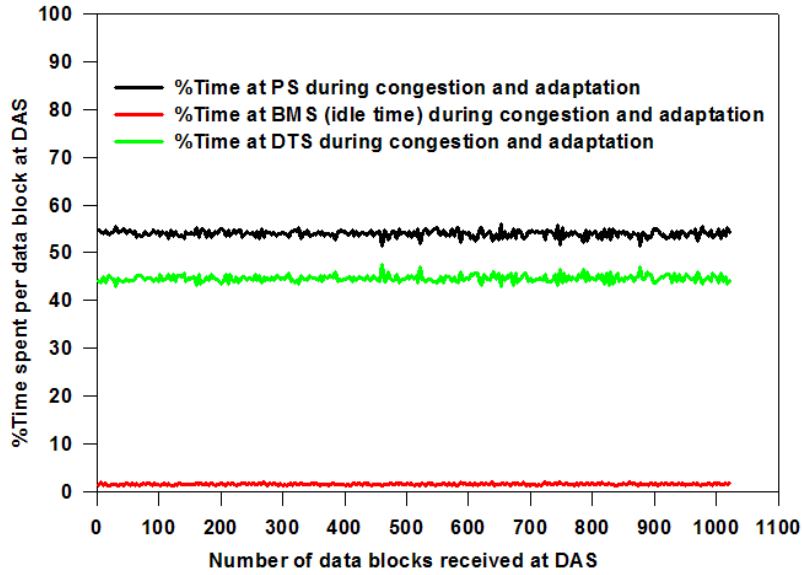

Figure 7.13: Breakup of %Time Spent at the Each of the Services Comprising the DAS per Data Block during Congestion with Adaptation

80%. Higher buffer occupancies at the in-transit nodes may lead to failures and result in data being dropped, and can impact the QoS at the sink. After the congestion clears and ADSS stops throttling data, the average %buffer occupancy at the in-transit nodes resets to around 50-57%.

7.3.5 Effect of Adaptations at In-Transit Nodes on the Quality of Data Received at Sink

This experiment measures the quality of data received at the sink, in terms of the number of processing functions completed, with congestion and with and without in-transit adaptations. The higher the number of processing functions completed, the higher the quality and utility of the data to the sink. The quality of data without adaptations is plotted in Figure 7.16. It can be seen from the plot that during congestion, the cumulative amount of data received at the sink with 3 processing functions (PS) applied is 0 MB, while the cumulative amount of data received with 2 processing functions (PS) applied is around 300 MB. In contrast, when adaptation

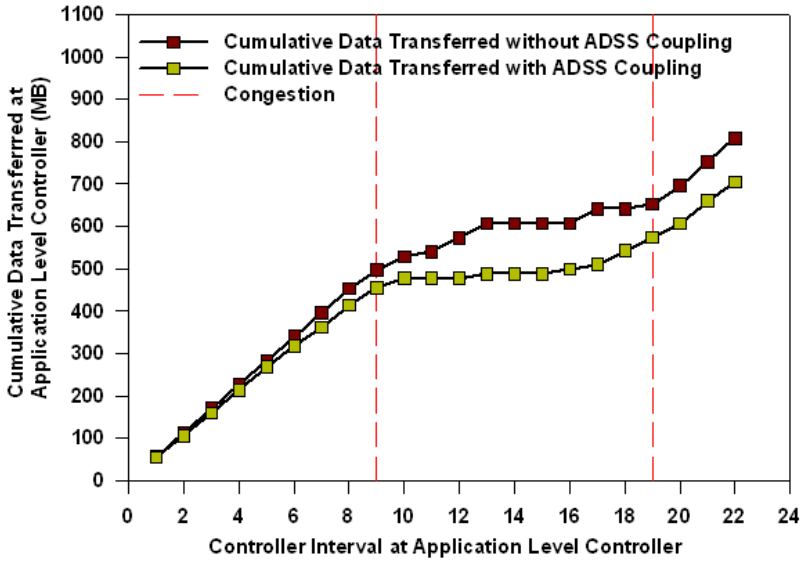

Figure 7.14: ADSS Behaviour with and without Coupling

at the in-transit nodes are turned on in Figure 7.17, the cumulative amount of data received at the sink with 3 processing functions (PS) applied is around 232 MB. Higher data quality can save significant time at the sink. For example, if the average processing time per data block is 1.6 sec, adaptations save about 372 sec (approx. 6 minutes) of processing time at the sink.

7.3.6 Effectiveness of End-to-End Cooperative Management

This experiment measures the cumulative amount of data that is not delivered on time to the sink with only application level management and with end-to-end cooperative management. This is plotted in Figure 7.18. In all cases, when there is no congestion, all data blocks reach the sink. However, when there is congestion, if only application level management is used, about 399 MB does not reach the sink. When cooperative management is used, this drops to around 294 MB.

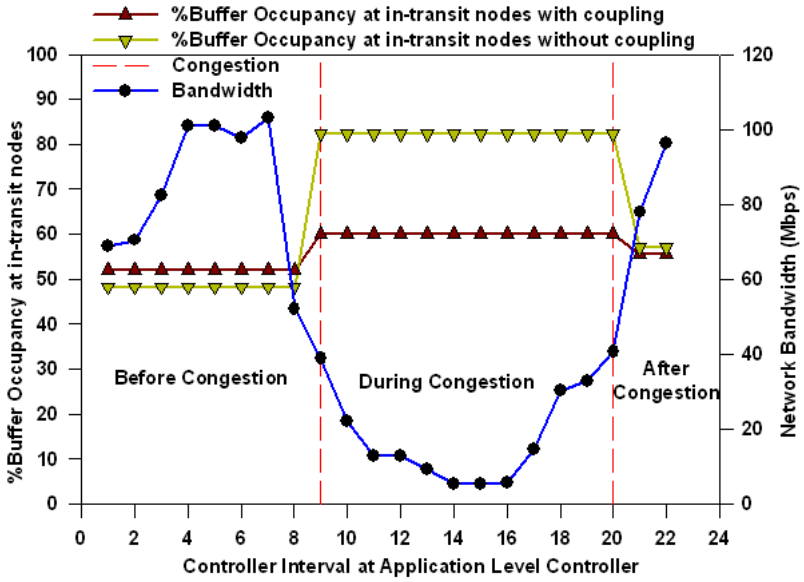

Figure 7.15: Average %Buffer Occupancy at the In-Transit Nodes with Coupling

7.4 Conclusion

This chapter presented the two level self-managing framework for in-transit manipulations of data in scientific workflows. The first level uses application level online controllers for high throughput data streaming while the second level of management operating at the in-transit nodes uses reactive strategy for processing data. It was found that this two level cooperative scheme achieves good QoS management for real-workflows involving the GTC application even during network congestions. In future we will investigate various strategies involving utility and micro-economic principles for scheduling in-transit computations in Grid workflows.

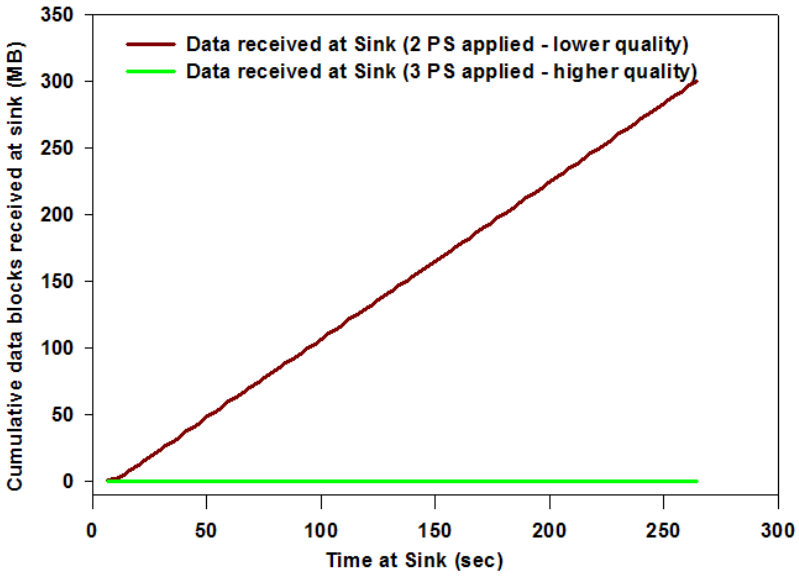

Figure 7.16: Quality of Data Received at Sink During Congestion without Adaptation at In-Transit Nodes

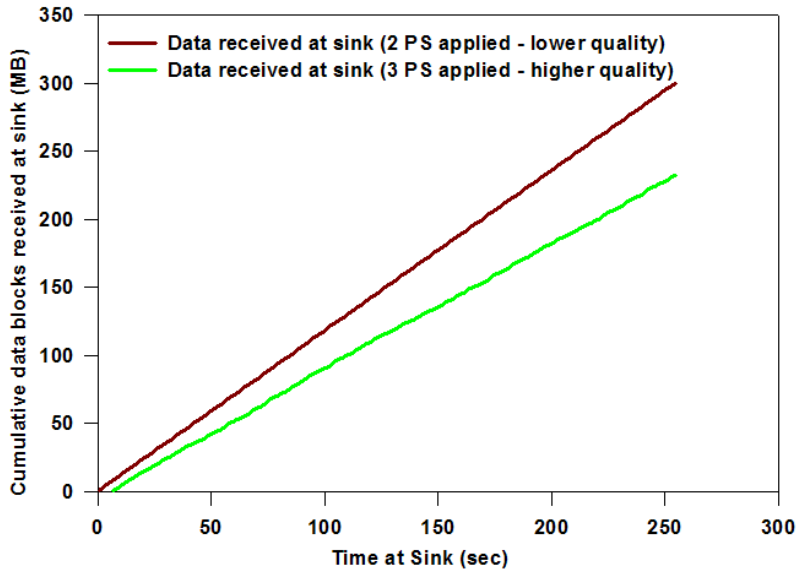

Figure 7.17: Quality of Data Received at Sink during Congestion with Adaptation at In-Transit Nodes

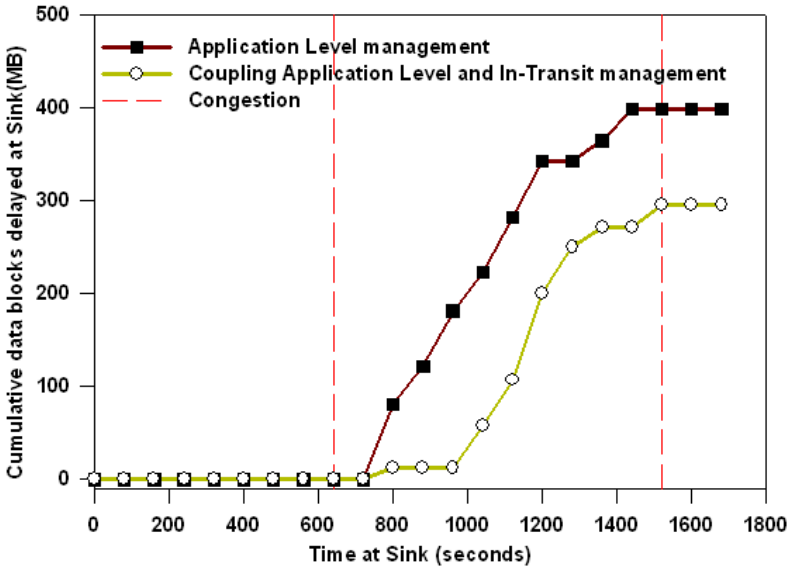

Figure 7.18: Cumulative Amount of Data that does not Reach the Sink In Time with and without Cooperative Management

Chapter 8

Slack-based Provisioning of In-Transit Processing for Data Intensive Scientific Workflows

8.1 Introduction

High-performance computing is playing an important role in science and engineering and is enabling highly accurate simulations, which provide insights into complex physical phenomena. However as computing systems grow in scale, complexity, and capability, effectively utilizing these platforms to achieve desired computational efficiency in both time and space becomes increasingly important and challenging. A key challenge is managing the enormous data volumes and high data rates associated with these applications, so as to have minimal impact on the execution of the simulations.

Furthermore emerging scientific and business application workflows are based on seamless interactions and coupling between multiple and potentially distributed computational, data and information services. This requires addressing the natural mismatches in the ways data is represented in different workflow components and on a variety of machines, and being able to "outsource" the required data manipulation and transformation operations to less expensive commodity resources "in-transit". Satisfying these requirements is challenging, especially in large-scale and highly dynamic in-transit environments with shared computing and communication resources, resource heterogeneity in terms of capability, capacity, and costs, and where application behaviors, needs, and performance are highly variable.

The overall goal of this research to develop a data streaming and in-transit data

scheduling and manipulations service that provides mechanisms as well as the management strategies for data intensive scientific workflows to addresses the requirements outlined above. In the previous chapter (Chapter 7) in-transit data manipulation and transformation was addressed using static resources in the the data path between the source and the destination.

In this chapter, slack metric based strategies are used to address the issue of scheduling and provisioning in-transit computations on an dynamic overlay of in-transit nodes. As discussed in Chapter 3 QoS objectives of both the Application level and In-Transit level are captured using a slack metric which bounds the time availability for data processing and transmission, such that the data reaches the sink or end-point in a timely manner. The in-transit nodes use the slack metric to make an optimum selection of resources in the dynamic overlay path and in turn minimize the end-to-end delay and maximize the quality of processed data in the overall workflow. The specific objectives of this chapter are (1) To capture QoS objectives at both application and in-transit levels using the concept of slack for in-transit data scheduling and manipulation (2) To investigate the coupling of slack metric strategies at the application level with slack managers at the in-transit level to create a cooperative management framework for data-intensive scientific workflows.

The rest of this chapter is organized as follows. Section 8.2 briefly presents the overall architecture of the cooperative management framework for data streaming and in-transit processing using the slack metric. Section 8.3 illustrates the design of the slack metric at the LLC controller. Section 8.4 discusses about the design and interaction of the in-transit nodes using the slack metric to adaptively process data generated at the application level. Section 8.5 couples both the in-transit and application level management to achieve QoS for the workflow. Section 8.6 describes the implementation of the slack based in-transit data manipulation framework for the Fusion Simulation Workflow (FSP). Section 8.7 presents preliminary results using the slack metric. Section 8.8 concludes the chapter.

8.2 A Self-Managing Service for Data Streaming and In-Transit Processing

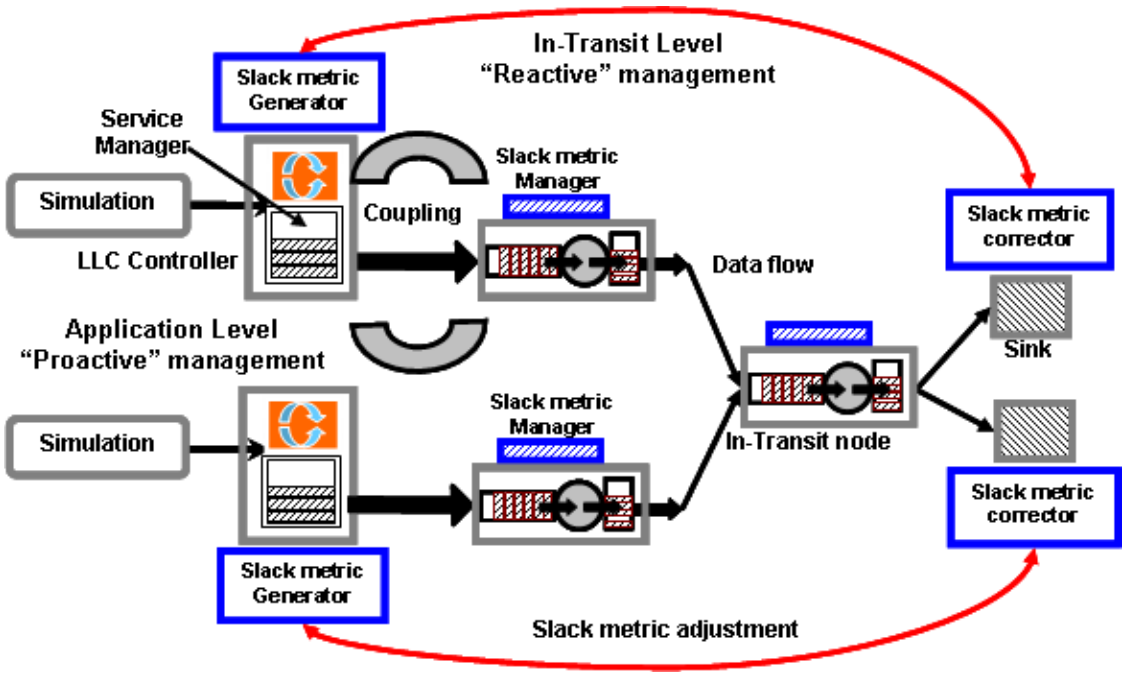

Figure 8.1: Self-managing Data Streaming and In-Transit Processing Service

A conceptual overview of the self-managing data streaming and in-transit processing service for data intensive scientific workflows is presented in Figure 8.1. It consists of two key components: The first is an application level data steaming service, which provides adaptive buffer management mechanisms and proactive QoS management strategies based on online control and user-defined polices, at application end-points. It also consists of slack generator at the LLC controller and slack corrector at end-points to ensure timely delivery of data. The second component provides scheduling mechanisms and adaptive runtime management strategies for in-transit data manipulation and transformation. Each in-transit node updates the slack metric on each data item. It also uses the slack metric to make an optimum selection of resources in the dynamic overlay path and in turn minimizes the end-to-end delay and maximizes the quality of processed data in the overall workflow. These two components

work cooperatively to address the overall end-to-end application constraints and QoS requirements outlined in Section 1.2. Some components in this service have been addressed in our previous Chapters 6, 7 and papers [16] [17] and is briefly summarized below. This chapter however focuses on the slack based provisioning and in-transit processing on dynamic overlays consisting of heterogeneous resources as well as their couplings with the application level mechanisms.

8.3 Application Level Data Streaming

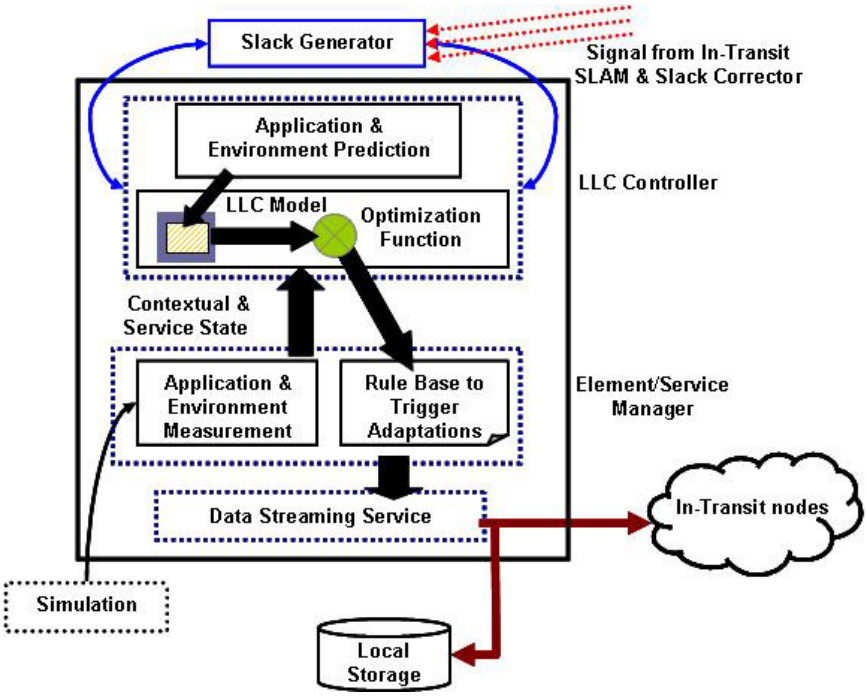

Figure 8.2: Application Level Data Streaming Service and Slack Generator

The application level self-managing data streaming service combines model-based limited look-ahead controllers (LLC) and rule-based autonomic managers with adaptive multi-threaded buffer management and data transport mechanisms at the application endpoints. It is constructed using the Accord-WS infrastructure for self-managing Grid services [53] and supports high throughput, low latency, robust application level data streaming in wide-area Grid environments as demonstrated in [16,

18]. The autonomic data streaming service is illustrated in Figure 8.2 and consists of a service manager and an LLC controller. The service manager monitors the state of the service and its execution context, collects and reports runtime information, and enforces the adaptation actions determined by its controller. Augmenting the element manager with an LLC controller allows human defined adaptation polices, which may be error-prone and incomplete, with mathematically sound models and optimization techniques for more robust self-management. Specifically, the controller decides when and how to adapt the application behavior and the service managers focus on enforcing these adaptations in a consistent and efficient manner. Additionally the slack generator at the application level uses input from the LLC controller to initially fix slack on data items generated. It also uses inputs received from the slack manager at the in-transit node (SLAM) and from the slack correctors at the sink to update (increase/decrease) the slack metric.

8.3.1 Slack Metric Generator

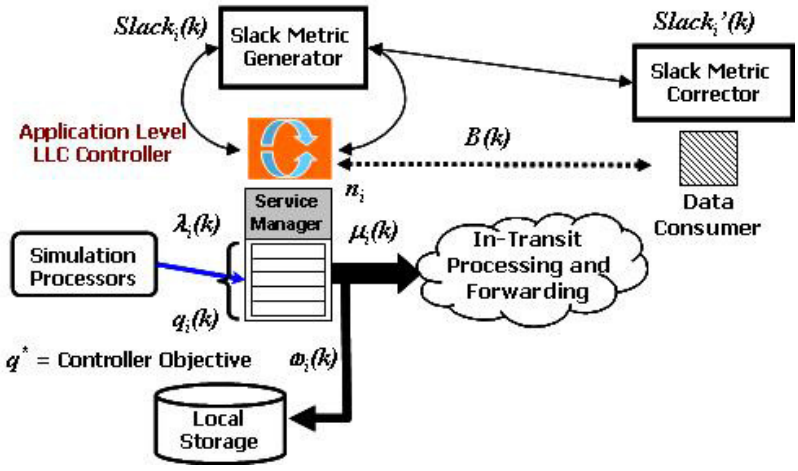

Figure 8.3: Design of the Slack Metric Generator for an Application Level Data Manipulation and Streaming

The structure of the LLC-based online controller is shown in Figure 8.3. The figure shows the key operating parameters for the controller at simulation node n_i at time

step k which are as follows. (1) State variable: The current average buffer size at n_i denoted as $q_i(k)$. (2) Environment variables: $\lambda_i(k)$ denotes the data generation rate into the buffer q_i and $B(k)$ the effective bandwidth of the network link from source to the sink. (3) Control or decision variables: Given the state and environment variables at time k, the controller decides $\omega_i(k)$ and $\mu_i(k)$, the data-transfer rate over the remote storage (Data Grid) and to the local storage respectively [16]. The objective of the controller denoted by q^* is to keep the %buffer occupancy $q_i(k)$ (%data blocks in the buffer) at zero. Note that $q_i(k)$ should be less than 100% or size of buffer so that the buffer does not overflow.

The LLC controller is augmented with the slack metric generator to minimize the execution and forwarding time in-transit while meeting a strict deadline of QoS requirements at the sink. Slack metric is the deadline fixed by the controller by which the processing and forwarding of the data must be completed in-transit before the data item reaches the sink. In doing so, it is guided by factors such as cost and speed of transferring, processing and queuing data items along the path. The slack at the LLC controller denoted as $Slack_i(k)$, is initially calculated based on the number ($db_p(k)$) of standard data blocks of size ($stdblock_p$) (for example 1MB, 2MB, 4MB) contained in the data items being transferred ($\mu_i(k) * q_i(k)$) over the network with bandwidth $B(k)$. It also includes the time taken to process each data block of standard size ($stdblock_p$) with its respective in-transit function $f_1..f_n$ (denoted as $t(f_j)$ at the sink (obtained through previously executing in-transit functions at the sink (history)). It is assumed here that the computational capacity at the sink is substantially lower than the in-transit nodes. The size of the each datablock is fixed depending on the application which generates the data. The slack calculation at each data streaming application level node n_i is as per the following equation:

$$Slack_i(k) = (\mu_i(k) * q_i(k))/B(k) + \sum_{p=1}^{n} db_p(k) * \sum_{j=1}^{n} t(f_j)$$

$$\forall db_p(k) = (\mu_i(k) * q_i(k))/(stdblock_p)$$

Each in-transit node updates the $Slack_i(k)$ as it is processed and forwarded by subtracting the time spent by the data item in-transit from $Slack_i(k)$. When the data item reaches the sink, it has a value $Slack'_i(k)$ and denotes if the data items generated at the application reached the sink on time. A negative value indicates that the data items reached the sink later than than the slack generator had estimated. A positive value indicates that the slack generator over-estimated the processing and forwarding requirements at the in-transit level. A feedback of this value will be reported to the slack generator for further action in the next controller cycle.

8.4 In-Transit Nodes

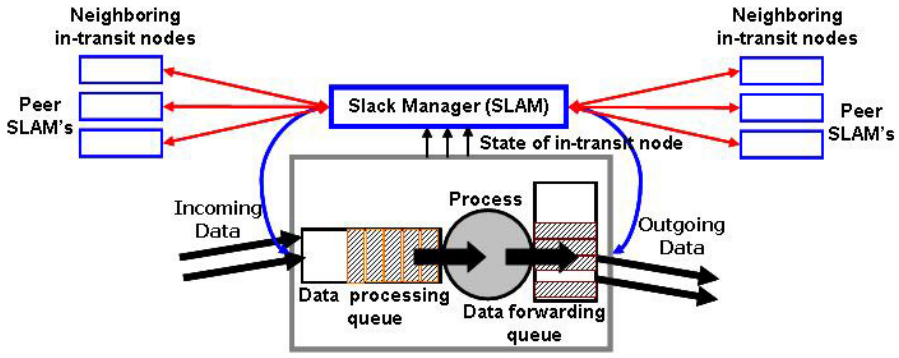

Figure 8.4: Architecture of an In-Transit Node

The in-transit data manipulation framework consists of a dynamic overlay of in-transit nodes which are on the path from the source to the destination. The in-transit node services run on small to medium commodity clusters with heterogeneous capabilities, loads and networks connecting them. These nodes are shared between multiple scientific workflows. The conceptual architecture of a node is illustrated in Figure 8.4. Each node performs the following steps during its normal operation, it first places the data it receives in a data processing queue for performing in-transit functions. The decision to forward or process the data is based on slack of the received

data item and various network or load conditions on the in-transit node. The decision of doing processing of a data item or forwarding it to the next hop is taken by the slack metric manager or SLAM. If the SLAM decides to process data items it does so by placing data on top of the data processing queue and invoking a processing thread. The processing thread applies the right in-transit function on it, by observing the metadata of the data. The amount of in-transit processing which has been done on the data item is logged back into the data item itself. The amount of data processing done in the in-transit node depends on the capacity and capability of the node and the available slack for the data item received. Negative or near 0 slack values result in data being immediately forwarded to the destination, while positive slack triggers further in-transit processing. The in-transit node then buffers the data items for forwarding to the next hop in the in-transit overlay or to the sink. The SLAM then updates the current slack ($Slack_i(k)$) on the data item based on the in-transit time which includes data processing, queuing and forwarding time. If the in-transit node cannot do a local processing on the data and the data has positive slack value, SLAM searches for the best in-transit node in its neighborhood that can satisfy its slack constraints on the data, in doing so it also takes into account the time required for sending the data item to the next hop. A processing that is not completed at the in-transit nodes has to be done at the sink. The current design of the framework assumes that each node can perform any of the required in-transit functions.

8.4.1 Adaptations at In-Transit Nodes

Adaptive Processing of Data at In-Transit Nodes

Congestion and overloading of an in-transit node can cause buffer overflows, loss of data and failure or delay of data reaching the sink. The slack metric managers (SLAM) at the in-transit nodes need to take corrective action to prevent such failures. As seen from Chapters 3 and 7, the time spent by a data item per in-transit node is

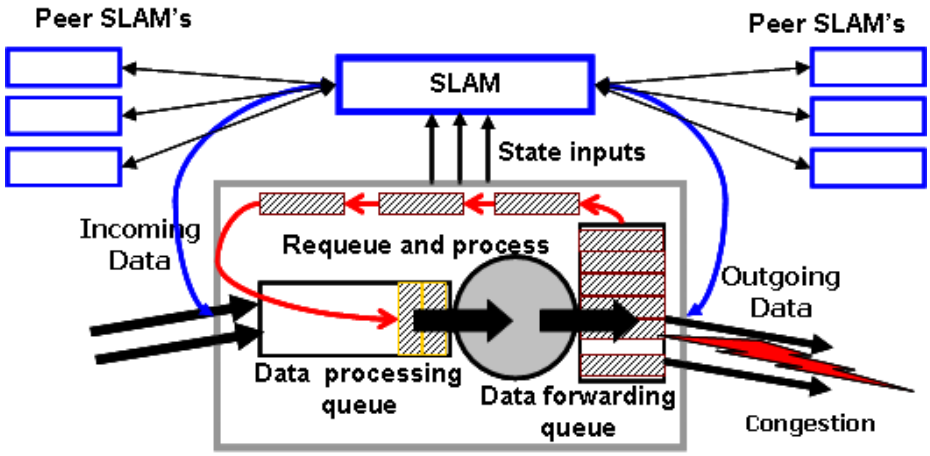

Figure 8.5: Adaptive Processing of Data at In-Transit Nodes by Re-Queuing

the sum of processing, buffering and forwarding times. During network congestions or increased loads downstream (due to overloading of data processing queues), buffering time significantly increases in relation to processing and forwarding times. If the buffer occupancy of current data forwarding queues reach a certain threshold and data processing queue is lightly loaded, SLAM adaptively re-queues data items to the data processing queue. This re-queuing is also based on the present slack value of the data item. This scenario is highlighted in Figure 8.5.

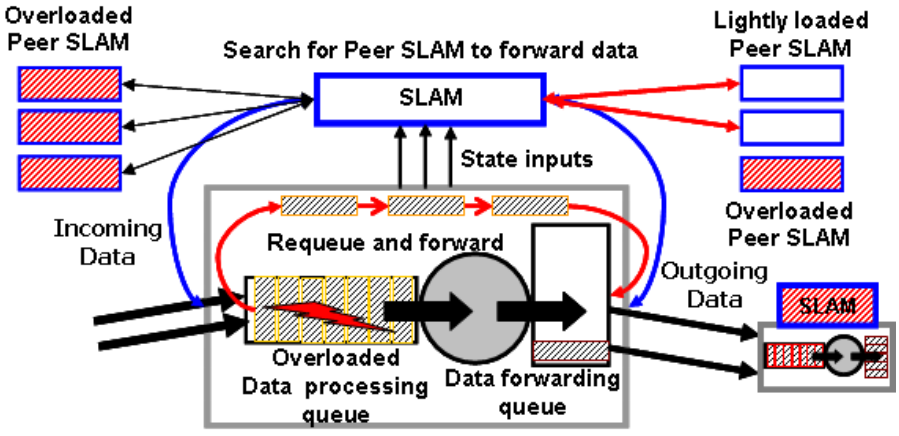

Figure 8.6: Adaptive Load Balancing of Data at In-Transit during Overloading

Adaptive Load Balancing of Data at In-Transit Nodes

Similarly during overloading of a in-transit node due to the execution of in-transit functions or other shared jobs, data in the data processing queue may suffer long waiting times. If data processing queues are saturated (i.e. node is heavily loaded) and forwarding queues are lightly loaded (downstream node(s) are lightly loaded and no congestion), SLAM re-queues data to forwarding queue instead of processing the data items (as illustrated in Figure 8.6). The data items from the processing queues are selected based on their slack metric. Data items with negative slack are prioritized over positive slack values, when selecting and re-queuing to the forwarding queues. Alternatively if the forwarding queues are also saturated (downstream node(s) are heavily loaded or there is congestion), SLAM first forwards data items to peer nodes in the overlay with the smallest median slack metric for data items in its data processing queue. Once the buffer occupancy in the forwarding queue is reduced, the SLAM then re-queues data from processing queue to the forwarding queue. This helps in minimizing impact of congestion/high loads on end-to-end performance.

8.5 Cooperative Self-management: Coupling Application Level and In-Transit Management

The application level and in-transit management can be coupled to achieve cooperative end-to-end self-management. Coupling is beneficial in cases of network congestion or CPU overload. In cases of network congestion data may not reach the sink and in cases CPU overload the in-transit node may become unavailable for in-transit processing. In the standalone case as illustrated in Figure 8.7, if the application level controller and the slack manager was used in isolation without feedback from the in-transit nodes and through the sink, the controller would detect a decrease of parameter $B(k)$. It would advise the service manager to increase $\omega_i(k)$ and decrease $\mu_i(k)$ to reduce the amount of data sent over the network. While this would eventually

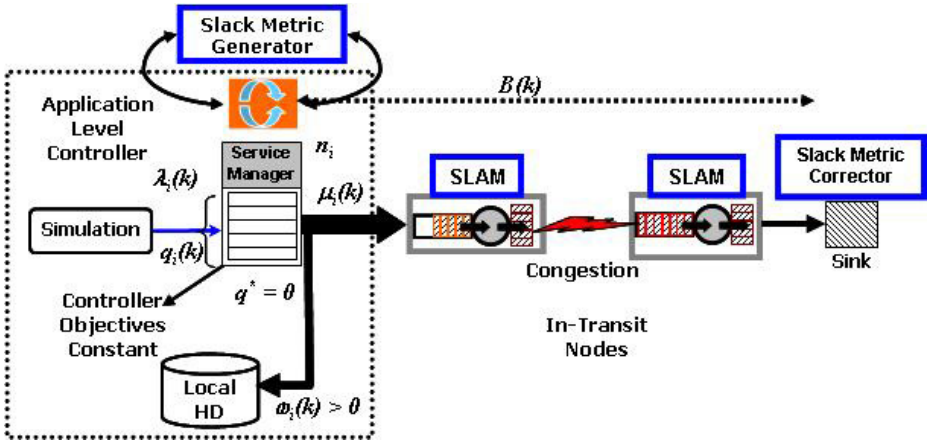

Figure 8.7: No Interaction between Application and In-Transit Level during Load Imbalance and Network Congestions at In-Transit Level

reduce network congestion, it would not compensate for the time required to manually transfer the data items once the simulation has completed and process it at the sink. It also does not take into account the nature of varying application QoS requirements and the possibility of potential CPU overloading and current state at the in-transit nodes. In the coupled scenario, there are two levels of interaction, which help the

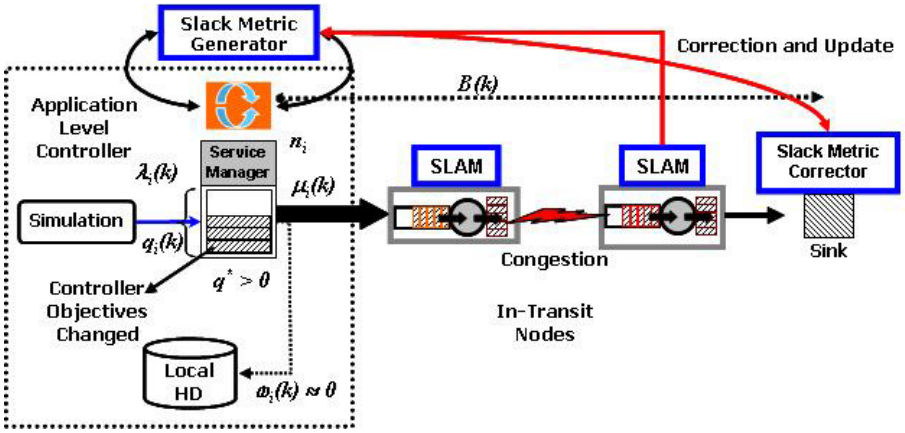

Figure 8.8: Interaction between Application and In-Transit Level during Load Imbalance and Network Congestions at In-Transit Level

application level controller to learn about events in the in-transit nodes. First in case of congestions at in-transit nodes, buffer occupancies of data forwarding queues at in-transit queues increase significantly. These in-transit nodes signal application level controllers about network congestion events. This allows the application level

controller for that data stream to detect the congestion more rapidly, rather than wait for the data item to reach the sink. In response to this event the application level controller increases $q_i(k)$ (or in turn q^* the controller objective) to a value higher than zero so as to throttle data items in the buffer.

Though this approach reduces the amount of data written to hard disk at the application level controller, it does not take into account the processing capacities and the current state at the in-transit node. The slack corrector at the sink observe the time taken for processing and forwarding at each in-transit node for a particular data stream. If they observe, that slack for a particular data item has exceeded more than the required threshold, they inform the application level controller, to increase the slack allocated for the data items by increasing the processing time for function $f_1..f_n$. Once the congestion clears, the slack correctors at the sink observe that the slack allocated is sufficient (slack metric has a positive value) and instruct slack generators at the application level controllers to decrease the slack metric per data item. This end-to-end and in-transit interaction by observing the state of the in-transit nodes, helps to achieve QoS under dynamic operating conditions for these workflows.

8.6 Implementation of the Framework for the Fusion Simulation Workflow

This section presents experiments using the cooperative self-managing data streaming service as a part of the fusion workflow. The overall application setup is shown in Figure 7.9. It consists of the Simulation Service (SS), i.e., the GTC fusion simulation, which runs at NERSC (CA) and ORNL (TN), and streams data for analysis to PPPL (NJ) and final data archiving at Rutgers University (NJ). The simulation service (SS) executes on 32 to 256 processors on "Jacquard" [65], at NERSC, and on 256 processors on "RAM", an SGI Altix machine. The Autonomic (self-managing) Data

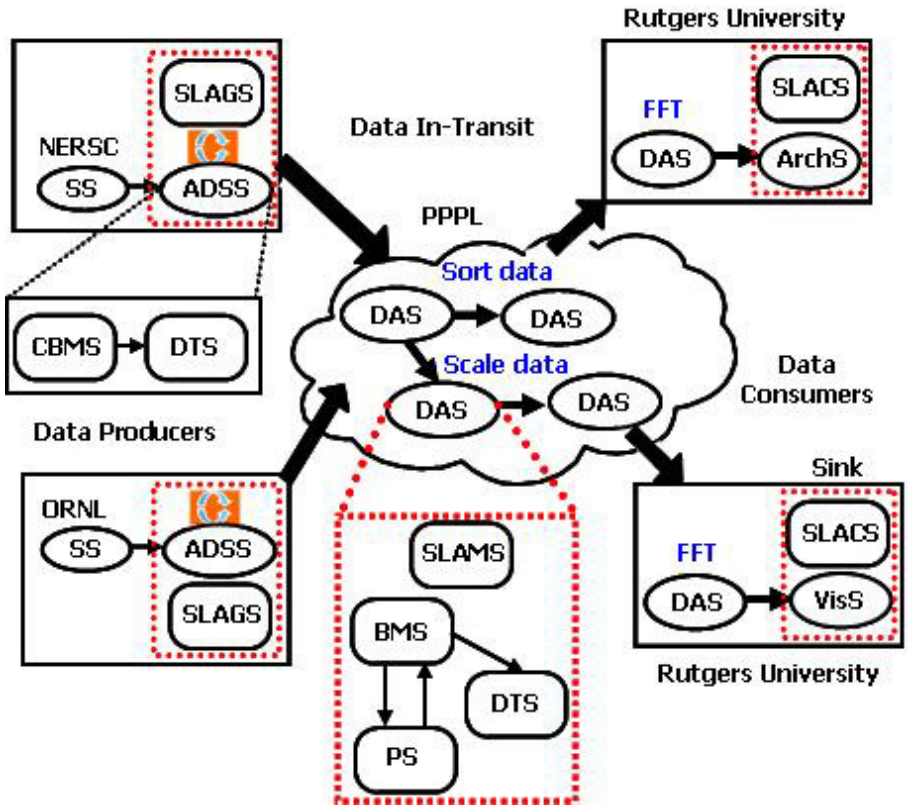

Figure 8.9: Slack based Fusion Simulation Workflow Implementation

Streaming Service (ADSS) is co-located with the SS at NERSC and ORNL. The in-transit processing is performed by the Data Analysis Service (DAS) located at the in-transit nodes at PPPL and Rutgers. Data Archiving Service (ArchS) is also located at Rutgers which is referred to as the sink or data consumer. Three in-transit nodes were used in these experiments. These included 32 AMD Athlon MP 2100+ processors ("gridn" cluster), 4 dual-core AMD Opteron processors ("portalx" cluster) both located at PPPL and a 64 processor Intel Pentium (1.70GHz) Beowulf cluster ("Frea") located at Rutgers. Note that there is a 155 Mbps (peak) ESNET [50] connection between PPPL and NERSC and a 100 Mbps network connection between PPPL and Rutgers.

The ADSS service consists of a Controller based Buffer Management Service (CBMS), which contains an LLC online controller, and a Data Transfer Service (DTS).

The controller interval for the CBMS was set to 80 seconds based on the data generation at the simulation end [16]. DTS uses a generic high performance transfer library for transferring data from simulation machines and is based on Logistical Networking (LN) [73]. The ADSS was enhanced with the SLAGS (Slack Generator Service) for generating slack metric on data items generated at the application. Similarly the sink was enhanced with SLACS (Slack Correction Service) for observing slack on data items received at sink and in-turn helps the SLAGS update slack metric value based on current in-transit network congestion and load.

The Data Analysis Service (DAS) operating at PPPL and Rutgers consists of the Processing Service (PS), Reactive Buffer Management Service (BMS), Data Transfer Service (DTS) and Slack Management Service (SLAMS) for managing the slack's of data items in the workflow. The DAS consumes data blocks streamed from the simulation or adjacent DAS services, and after applying the requisite PS it forwards them to the appropriate DAS based on the decision of the SLAMS. Three in-transit processing functions were used in these experiments, they included sorting, scaling and FFT, each of which could be run on any of the in-transit nodes. The experiments conducted are presented in the next section.

8.7 Evaluation

This section presents preliminary results obtained using the slack metric. It first discusses the construction of the slack metric using the input from the LLC controller and from history values obtained by previously executing the in-transit functions on the sink.

8.7.1 Benchmarking In-Transit Functions

This section studies the performance of three in-transit function which include Data Scaling("scale"), Quicksort("qsort") and Fast Fourier Transform("fft") over three different infrastructures which include the source, in-transit overlay and sink resources. These functions were first executed on lightly loaded machines for various data sizes ranging from 1MB to 1024 MB depending on the in-transit functions. Later these functions were executed on the sink machine which had load simulator programs running. The purpose of this experiment was to derive slack values at the slack metric generator in cases of congestion at the in-transit overlay resources. The idea is to increase the value of the slack till they reach the maximum possible value allowed by the system. Figure 8.10 illustrates that qsort time at the source for 4MB data is .69

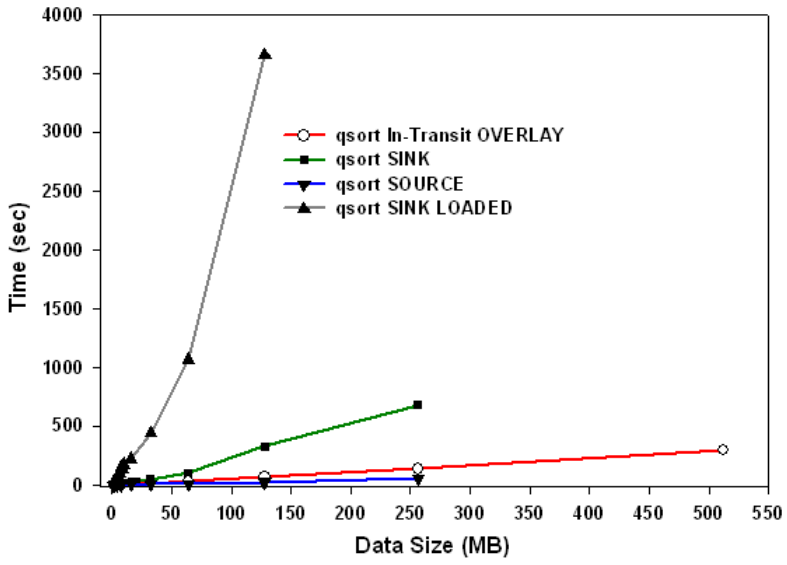

Figure 8.10: Benchmarking "qsort" In-Transit function for Deriving Slack Metric

sec., while the same 4MB data requires 1.5 sec. at the overlay nodes and 5.5 sec. at the sink. But after the sinks are loaded the qsort function suffer and require 57 sec. to complete. When the sinks are loaded they can also process data till around 128MB due to memory available for processing is significantly reduced. For example 128MB takes 3564 sec./$\approx$ 60 minutes to process at the sink. Hence it is essential to execute

in-transit functions in the overlay for large data sizes making in-transit processing attractive for end-to-end workflow. As seen from Figure 8.11 when running the fft

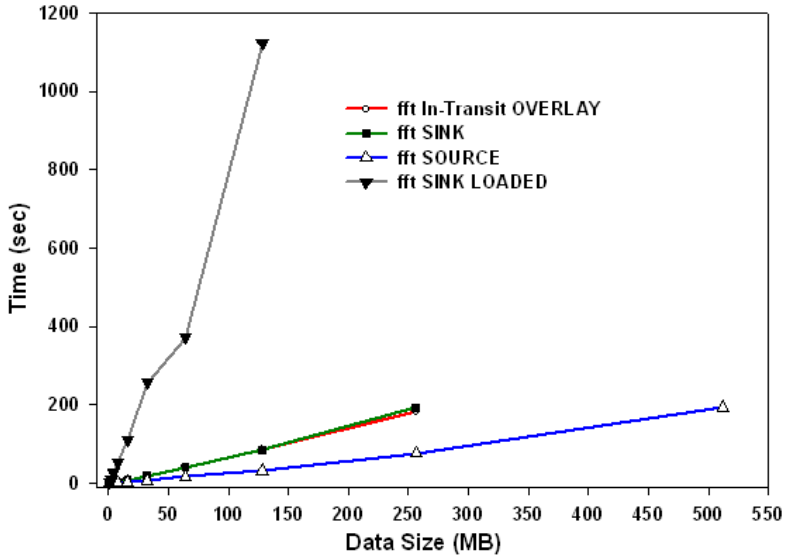

Figure 8.11: Benchmarking "fft" In-Transit function for Deriving Slack Metric

benchmark the time to execute this function on both the in-transit overlay and sink are nearly equivalent, as the function does not consume much memory and is not recursive as compared to the qsort function. The time to execute the fft function on the source is slightly lesser compared to both the in-transit and sink, but the source is able to process data items of size 512MB as compared to the sink/overlay nodes. When the sink is loaded, it takes around 1122 sec./$\approx$ 20 minutes to run fft function on 128MB. Finally we benchmark the scale function all resources, and we observe that there is a linear scaling with increasing data sizes even with loading at the sink. We also observe that there is a breakdown at the sink when the scale function executes on 456MB of data, due to memory related issues. Similarly a loaded sink cannot execute scale functions on data sizes above 128MB in a reasonable amount of time.

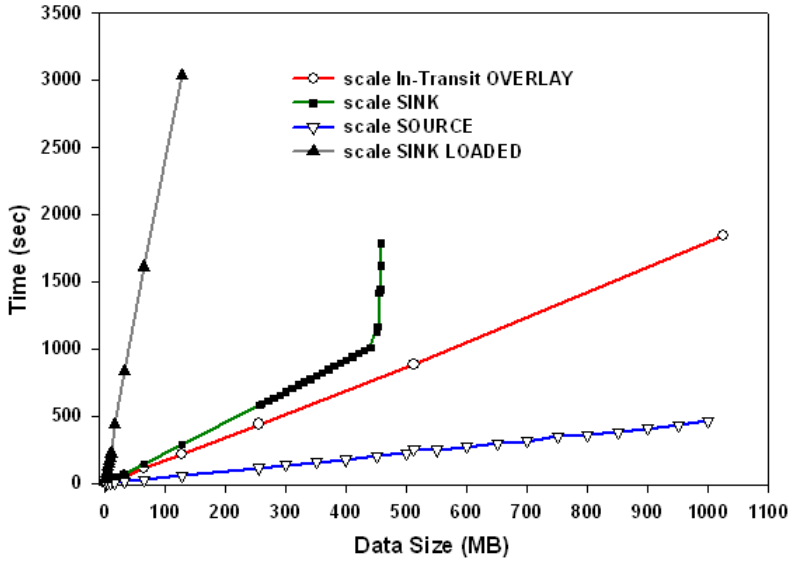

Figure 8.12: Benchmarking "scale" In-Transit function for Deriving Slack Metric

8.7.2 Benchmarking Forwarding Time

To benchmark the end-to-end forwarding time data was sent from source to the sink via the in-transit overlay, additionally data was sent among the in-transit nodes. From Figure 8.13 it is observed that end-to-end forwarding time was significantly higher due to lower buffering capacity at the sink. For example 4MB of data required 0.2 sec. for forwarding in the overlay while the same data took 6 sec. for end-to-end forwarding. Hence if the in-transit nodes are loaded and buffering capacity is limited, it incurs lesser overhead to forward data items to nearby in-transit nodes for load-balancing.

8.8 Conclusion

This chapter presented the two level self-managing framework using a slack metric for in-transit manipulations of data in scientific workflows. The first level operating at application level uses proactive management strategies for data streaming and generates the slack metric for capturing QoS of the application data at both levels and ensures strict time deadlines are met at the sink. The second level of management

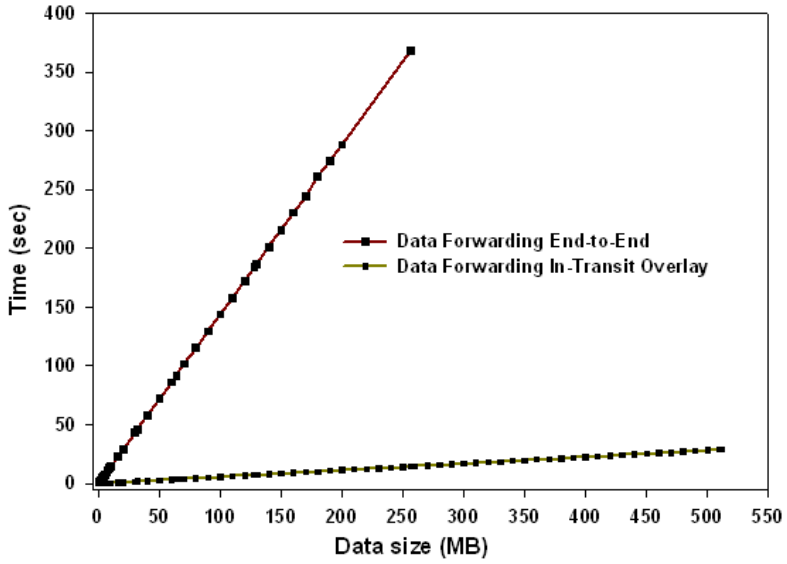

Figure 8.13: Benchmarking Data Forwarding Time both End-to-End and Within the In-Transit Overlay

operating at the in-transit nodes uses opportunistic strategies for processing data using the slack metric as a guiding parameter. Initial experiments were conducted to generate slack parameter at the application level using benchmarking techniques. In future we plan to conduct extensive experiments on Emulab [35] infrastructure to study the effects of slack metric on in-transit processing by creating interesting overlay topologies. We also plan to evaluate methods to trade off slack for maximizing quality of processed data reaching the sink.

Chapter 9
Conclusions and Future Work

This chapter presents the summary of the thesis and the future work to be conducted.

9.1 Summary

This thesis presented the design and implementation of a self-managing data streaming and in-transit processing service that enables efficient data transport and manipulation to support emerging Grid-based scientific workflows. The presented design of end-to-end QoS management combines rule-based heuristic adaptations with more formal model-based online control strategies to provide a self-managing service framework that is robust and flexible, and can address the dynamism in application requirements and system state. The fusion simulation workflow was used to evaluate the data-streaming and in-transit processing service and its self-managing behaviours. The results demonstrate the ability of the service to meet Grid-based data-streaming requirements, as well as its efficiency and performance. The work completed to date is summarized below.

- **End-to-End Self Management Mechanisms:**
  - **Adaptive Buffer Management strategies:** Buffer management using simple strategies was used as an initial framework for deploying end-to-end data streaming applications.
  - **Self-Managing Data Streaming using Accord:** Policy and rule based programming framework was used for inducing adaptive behaviors into the

data streaming framework, taking into account the key characteristics of Grid execution environments.

  - **Self-Managing Data Streaming using Rule and Model based Online Control:** Advanced control formulations offered a theoretical basis for self-managing adaptations in distributed applications. As a result a combination of typical rule-based self-management approaches with formal model-based online control strategies was used to introduce adaptive behaviour in data streaming applications.

- **Reactive and Opportunistic Mechanisms for In-transit Data Manipulation:** Quick reactive strategies operate at in-transit processing elements which process data items from multiple streams.

- **Slack-based Provisioning and In-Transit Data processing:** to make an optimum selection of resources in the dynamic overlay path and in turn minimize the end-to-end delay and maximize the quality of processed data in the overall workflow.

- **Infrastructure and Deployment:** The wide area data streaming framework was deployed and operational for transferring data from NERSC to PPPL and from ORNL to PPPL. The testbed also allows the scientist perform data in-transit manipulations from NERSC/ORNL to Rutgers via PPPL clusters.

9.2 Future Work

Self-managing data streaming and in-transit processing framework was integrated in several scientific applications proving that it is a valuable component for realizing large-scale decentralized Grid workflows. Future workflow domains to be investigated include financial data streaming. With the popularity of technologies such as GPGPU's efforts will be made to integrate GPGPU's into the in-transit overlay which

presently use commodity clusters for data processing. Similarly technologies such as virtualization now makes it possible to enable efficient utilization of in-transit nodes. Additionally the two level self-managing framework will incorporate learning methods at the application and in-transit levels to better understand network and system behaviors, to tolerate overloads and system faults.

9.2.1 Study End-to-End Self-Management Mechanisms using Finite State Machines (FSM)

Model based online controllers are designed for stable operating parameters and are not suited for in-transit application workflows, where environment conditions or operating parameters, such as network state and CPU load, change rapidly. The end-to-end self management mechanisms in these cases need to adapt quickly. To address these issues, we will investigate self management mechanisms using a finite state machine approach. Each state of the finite state machine is associated with a feedback controller that is customized to that state. Each feedback controller has a small gain factor and is invoked in response to state transitions which occur due to frequent and asynchronous environment events. Unlike our previous approach of model based online control it is not necessary to construct models especially, in cases where models are difficult to build and verify. We anticipate that this new end-to-end self-management approach has a low overhead and works well at the data production ends, particularly in cases of data in-transit application workflows.

9.2.2 Incorporate Learning Methods at Application and In-Transit Levels

Self-managing data streaming applications in our framework are based on models that are constructed offline by observing application characteristics and behaviour. However when applications need to be reprogrammed or changed, they tend to exhibit

varied and unpredictable behaviour and this may lead to a significant re-modeling effort. We intend to use a learning loop (reinforcement learning (RL) techniques) to increase the accuracy of the models, by adjusting the model based on the previous mismatches between the model and performance of the application. Additionally learning could be used at in-transit level to learn changes in processing times of in-transit jobs and take corrective action on slack metrics. For this purpose it is intended to use the winGamma [33] software from University of Cardiff to analyze data from slack metric logs at in-transit nodes and also at the application level data streaming models to better understand data streaming application and operating environment behaviour. winGamma, is designed for non-linear data analysis (using the Gamma test) and non-linear modeling (using neural networks and local-linear regression).

9.2.3 Application to Financial Data-Streaming

Financial data comprising of individual stock, bond, securities and currency trades can be accumulated from multiple sources over the internet to produce massive data streams. Additionally sophisticated data processing engines such as the Bloomberg Terminal [56] (from Bloomberg L.P.) evaluate queries over real-time streaming financial data such as stock tickers and news feeds to produce meaningful data to traders and market analysts. Furthermore these devices can push market data over the internet to users on lightweight clients. Moreover consolidation of servers into data centers or server farms where data is stored, can enable traders to have access to large amount of data. All of these point to the fact that there is an increased data management challenge when dealing with real-world financial applications that require streaming application workflows. We intend to address these challenges by incorporating financial applications into the self-managing framework designed in this thesis. To solve the problem of data streaming and in-transit processing for financial applications several key assumptions need to be modified, one being that in-transit functions do not

change the size of the data. If these assumptions are removed from the problem formulation new strategies could be devised at the in-transit nodes, wherein in-transit functions which reduce the size of the data could be used to offset latency due to network congestions. Simultaneously the in-transit nodes and end-point needs to deal with issues where in-transit functions could produce more data than was input to the functions. Overall financial data streaming applications presents significant challenges and changes need to be made to the design of the slack metric at the LLC controllers to incorporate financial data into the self-managing framework.

9.2.4 Integration with GPGPUs into the In-Transit Overlay

The general-purpose GPU (GPGPU or GP^{2}2U) [37] computing phenomenon has gained momentum over the last few years, and has reached the point where it has been accepted as an application acceleration technique. Various innovative uses of GPUs include computing game physics between frames, linear algebra (e.g., LU decomposition), in-situ signal and image processing, database "SELECT" processing, finite element and partial differential equation solvers, and tomography image reconstruction, to name a few. Applications continue to appear on the horizon that exploit the GPU's parallelism and vector capabilities. In the future in-transit nodes in our scientific workflow could be replaced with GPGPU's to enable specialized processing. To account for this change we need to devise strategies at the in-transit level to take advantages of the parallel stream processing capabilities of GPGPU's. These strategies could involve exploiting the parallelism of these operations for scheduling jobs at GPGPU's and thus reducing overhead on in-transit nodes. Furthermore, we need to study if adaptive forwarding mechanisms at GPGPU's produce overhead on executing stream applications.

9.2.5 Virtualization of In-Transit Nodes

Virtualization [87] technology basically lets one computer do the job of multiple computers, by sharing the resources of a single computer across multiple environments. Essentially commodity clusters used in our in-transit processing node will host multiple in-transit processing units, freeing the in-transit processing from limitations of the underlying operating system. In addition the application of virtualization at in-transit nodes could lead to potential energy savings and lower capital expenses due to more efficient use of hardware resources. This can lead to better in-transit management, increased security across multiple application workflows, and improved disaster recovery processes. Virtualization technologies at in-transit nodes would enable us to spawn multiple instances of in-transit processing elements in a new virtual machine(vm) in case of increased load on the current in-transit elements. Though it needs to be seen if creating a new virtual machine, introduces significant overhead on the executing system. Additionally it could enable creation of multiple data streaming elements to allow for faster forwarding of in-transit data. Furthermore this technology introduces challenges at the in-transit management layer, where the slack manager need to deal with scheduling data processing efficiently across multiple vm's.

References

[1] S. Abdelwahed, N. Kandasamy, and S. Neema. A Control-Based Framework for Self-Managing Distributed Computing Systems. In *Workshop on Self-Managed Systems (WOSS'04)*, Newport Beach,CA USA, 2004.

[2] S. Abdelwahed, N. Kandasamy, and S. Neema. Online Control for Self-Management in Computing Systems. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 368–376, Le Royal Meridien, King Edward, Toronto, Canada, 2004.

[3] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance Guarantees for Web Server End-Systems: A Control Theoretic Approach. *IEEE Transactions on Parallel & Distributed Systems*, 13(1):80–96, 2002.

[4] A. Abrahams, D. Eyers, and J. Bacon. An Asynchronous Rule-Based Approach for Business Process Automation Using Obligations. In *Third ACM SIGPLAN Workshop on Rule-Based Programming (RULE'02)*, pages 323–345, Pittsburgh, PA, 2002. ACM Press.

[5] Manish Agarwal and Manish Parashar. Enabling Autonomic Compositions in Grid Environments. In *Fourth International Workshop on Grid Computing (Grid '03)*, pages 34–41, Phoenix, Arizona, USA, 2003. IEEE Computer Society.

[6] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing*, 28(5):749–771, 2002.

[7] W. Allcock, J. Bresnahan, R. Kettimuthu, and J. Link. The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid. In *Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models (HIPS-HPGC 2005)*, Denver, Colorado, USA, 2005.

[8] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped Gridftp Framework and Server. In *Super Computing (SC'05)*, pages 54–64, Seattle, WA, USA, 2005. IEEE Computer Society.

[9] I. Altintas, S. Bhagwanani, D. Buttler, S. Chandra, Z. Cheng, M. A. Coleman, T. Critchlow, A. Gupta, W. Han, L. Liu, B. Ludäscher, C. Pu, R. Moore, A. Shoshani, and M. Vouk. Modeling and Execution Environment for Distributed

Scientific Workflows. In *15th International Conference on Scientific and Statistical Database Management (SSDBM03)*, pages 247–250, Boston, Massachusetts, USA, 2003.

[10] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Load Management and High Availability in the Medusa Distributed Stream Processing System. In *ACM SIGMOD international conference on Management of data (SIGMOD '04)*, pages 929–930, Paris, France, 2004. ACM Press.

[11] M. Beck, T. Moore, and J. S. Plank. An End-to-End Approach to Globally Scalable Network Storage. In *ACM SIGCOMM '02*, pages 339–346, Pittsburgh, Pennsylvania, USA, 2002. ACM Press.

[12] F. Bertrand. The MxN problem in Distributed Scientific Computing. `http://www.cs.indiana.edu/~febertra/mxn/index.html`, 2004.

[13] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Nashua, NH, USA, 3 edition, 2005.

[14] M. Beynon, C. Chang, U. Catalyurek, T. Kurc, A. Sussman, R. Andrade, H.and Ferreira, and J. Saltz. Processing Large-Scale Multi-Dimensional Data in Parallel and Distributed Environments. *Parallel data-intensive algorithms and applications*, 28(5):827–859, 2002.

[15] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar. High Performance Threaded Data Streaming for Large Scale Simulations. In *5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, pages 243–250, Pittsburgh, PA, USA, 2004.

[16] V. Bhat, M. Parashar, M. Khandekar, N. Kandasamy, and S. Klasky. A Self-Managing Wide-Area Data Streaming Service using Model-based Online Control. In *7th IEEE International Conference on Grid Computing (Grid 2006)*, pages 176–183, Barcelona, Spain, 2006. IEEE Computer Society.

[17] V. Bhat, M. Parashar, and S. Klasky. Experiments with In-Transit Processing for Data Intensive Grid workflows. In *8th IEEE International Conference on Grid Computing (Grid 2007)*, pages 193–200, Austin, TX, USA, 2007. IEEE Computer Society.

[18] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, and S. Abdelwahed. Enabling Self-Managing Applications using Model-based Online Control Strategies. In *3rd IEEE International Conference on Autonomic Computing*, pages 15–24, Dublin, Ireland, 2006.

[19] T.E. Bihari and K. Schwan. Dynamic Adaptation of Real-Time Software. *ACM Transactions on Computer Systems*, 9(2):143–174, 1991.

[20] B. Biornstad, C. Pautasso, and G. Alonso. Control the Flow: How to Safely Compose Streaming Services into Business Processes. In *IEEE International Conference on Services Computing (SCC'06)*, pages 206–213, Chicago, USA, 2006. IEEE Computer Society.

[21] P. A. Buhler and J. M. Vidal. Towards Adaptive Workflow Enactment Using Multiagent Systems. *Information Technology and Management*, 6(1):61–87, 2005.

[22] C. Buragohain, D. Agrawal, and S. Suri. A Game Theoretic Framework for Incentives in P2P Systems. In *3rd International Conference on Peer-to-Peer Computing-P2P '03*, pages 48–56, Linkping, Sweden, 2003. IEEE Computer Society.

[23] L. Capra, W. Emmerich, and C. Mascolo. A Micro-Economic Approach to Conflict Resolution in Mobile Computing. In *Workshop on Self-healing Systems (SIGSOFT'02)*, pages 31–40, Charleston, SC, USA, 2002.

[24] A. Cervin, J. Eker, B. Bernhardsson, and K. Arzen. Feedback-Feedforward Scheduling of Control Tasks. *Real-Time Systems*, 23(1-2):25 – 53, 2002.

[25] J. Chen. M3D Home. `http://w3.pppl.gov/m3d/index.php`, 2007.

[26] J.J. Cheng, D. Flaxer, and S. Kapoor. RuleBAM: A Rule-Based Framework for Business Activity Management. In *IEEE International Conference on Services Computing(SCC'04)*, pages 262–270, Shanghai, China, 2004.

[27] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. In *Network Storage Symposium (NetStore '99)*, volume 23, pages 187–200, Seattle, WA, USA, 1999. Journal of Network and Computer Applications.

[28] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf. Performance and Scalability of a Replica Location Service. In *13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 182–191, Honolulu, Hawaii, USA, 2004. IEEE Computer Society.

[29] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 15 March 2001.

[30] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: Mapping Scientific Workflows onto the Grid. In *2nd European Across GRIDS Conference-AxGrids04*, volume 3165/2004 of *Lecture Notes in Computer Science*, pages 11–20, Nicosia, Cyprus, 2004. Springer Berlin / Heidelberg.

[31] P. Dickens, W. Gropp, and P. Woodward. High Performance Wide Area Data Transfers Over High Performance Networks. In *International Parallel and Distributed Processing Symposium (IPDPS'02)*, pages 254–262, Fort Lauderdale, Florida, USA, 2002.

[32] J. Ding, J. Huang, M. Beck, S. Liu, T. Moore, and S. Soltesz. Remote Visualization by Browsing Image Based Databases with Logistical Networking. In *ACM/IEEE conference on Supercomputing (SC'03)*, pages 34–44, Phoenix, Arizona, USA, 2003. IEEE Computer Society.

[33] P. Durrant, S. Margetts, and A.J. Jones. The winGamma User Guide. `http://users.cs.cf.ac.uk/Antonia.J.Jones/GammaArchive/winGammaManual/`, 2001.

[34] M.A. Eriksen. Trickle: A Userland Bandwidth Shaper for Unix-like Systems. In *USENIX Annual Technical Conference (USENIX'05)*, pages 61–70, Anaheim, CA, USA, 2005. SAGE.

[35] Flux-Research-Group. Emulab - Network Emulation Testbed. `http://www.emulab.net`, 2006.

[36] E. W. Fulp, M. Ott, D. Reininger, and D. S. Reeves. Paying for QoS: An Optimal Distributed Algorithm for Pricing Network Resources. In *Sixth International Workshop on Quality of Service (IWQoS'98)*, pages 75–84, Napa, CA, USA, 1998.

[37] GPGPU.org. GPGPU.org Wiki. `http://www.gpgpu.org/w/index.php`, 2008.

[38] Y. Gu and R.L. Grossman. SABUL: A Transport Protocol for Grid Computing. *Journal of Grid Computing*, 1(4):377–386, 2003.

[39] A. Hanushevsky. bbcp peer to peer cp program. `http://www.slac.stanford.edu/~abh/bbcp/`, 2002.

[40] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, Hoboken, NJ, 2004.

[41] J. L. Hellerstein, Y. Diao, and S. S. Parekh. Applying Control Theory to Computing Systems. Technical report, IBM Research Report, RC23459 (W0412-008), 7 December 2004.

[42] J. In, P. Avery, R. Cavanaugh, L. Chitnis, M. Kulkarni, and S. Ranka. SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 05)*, page 12.2, Denver, Colorado, USA, 2005. IEEE Computer Society.

[43] D. Jadav, A. N. Choudhary, and P. B. Berra. Techniques for Increasing the Stream Capacity of A High-Performance Multimedia Server. *IEEE Transactions on Knowledge and Data Engineering*, 11(2):284–302, 1999.

[44] N. Kandasamy, S. Abdelwahed, and J. P. Hayes. Self-Optimization in Computer Systems via Online Control: Application to Power Management. In *1st IEEE International Conference on Autonomic Computing (ICAC'04)*, pages 54–61, New York, NY, USA, 2004.

[45] M. Khandekar, N. Kandasamy, S. Abdelwahed, and G. Sharp. A control-based framework for self-managing computing systems. *Multiagent and Grid Systems, an International Journal*, 1(2):63 – 72, 2005.

[46] S. Klasky, M. Beck, V. Bhat, E. Feibush, B. Ludäscher, M. Parashar, A. Shoshani, D. Silver, and M. Vouk. Data management on the fusion computational pipeline. *Journal of Physics: Conference Series*, 16(2005):510–520, 2005.

[47] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, and R. Samtaney. Grid-Based Parallel Data Streaming implemented for the Gyrokinetic Toroidal Code. In *Supercomputing Conference (SC 2003)*, volume 24, Phoenix, AZ, USA, 2003.

[48] S. Klasky, B. Ludäscher, and M. Parashar. The Center for Plasma Edge Simulation Workflow Requirements. In *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, volume 00, page 73, Atlanta, GA, USA, 2006. IEEE Computer Society.

[49] T. Kosar, G. Kola, and M. Livny. Building Reliable and Efficient Data Transfer and Processing Pipelines. *Concurrency and Computation: Practice & Experience*, 18(6):609–620, 2006.

[50] Lawrence Berkeley National Laboratory. Energy Sciences Network (ESNET-4). `http://www.es.net/`, 2006.

[51] G. von Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. GridAnt: A Client-Controllable Grid Workflow System. In *37th Hawaii International Conference on System Science (HICSS'04)*, page 10, Waikoloa, Big Island, Hawaii, 2004.

[52] Z. Lin, T. S. Hahm, W. W. Lee, W. M. Tang, and R. B. White. Turbulent Transport Reduction by Zonal Flows: Massively Parallel Simulations. *Science*, 281(5384):1835–1837, 1998.

[53] H. Liu. *Accord: A Programming System for Autonomic Self-Managing Applications*. PhD thesis, Rutgers University, 2005.

[54] H. Liu, V. Bhat, M. Parashar, and S. Klasky. An Autonomic Service Architecture for Self-Managing Grid Applications. In *6th International Workshop on Grid Computing (Grid 2005)*, pages 132–139, Seattle, WA, USA, 2005.

[55] H. Liu, M. Parashar, and S. Hariri. A Component-based Programming Framework for Autonomic Applications. In *1st IEEE International Conference on Autonomic Computing (ICAC-04)*, pages 10–17, New York, NY, USA, 2004.

[56] Bloomberg L.P. Bloomberg Professional. `http://about.bloomberg.com/`, 2008.

[57] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online Data Migration with Performance Guarantees. In *USENIX Conference on File Storage Technologies (FAST'02)*, pages 219–230, Monterey, CA, 2002.

[58] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. *Real-Time Systems*, 23(1-2):85–126, 2002.

[59] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-Theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads. In *International Conference on Compilers, Architectures, & Synthesis Embedded Systems (CASES)*, pages 156–163, Grenoble, France, 2002. ACM Press.

[60] 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 Kepler System. *Concurrency and Computation: Practice & Experience*, 18(10):1039 –1065, 2006.

[61] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[62] X. Ma. *Hiding Periodic I/O Costs in Parallel Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 2003.

[63] X. Ma, J. Lee, and M. Winslett. High-Level Buffering for Hiding Periodic Output Cost in Scientific Simulations. *IEEE Transactions Parallel Distributed Systems*, 17(3):193–204, 2006.

[64] S. Mascolo. Classical Control Theory for Congestion Avoidance in High-Speed Internet. In *38th IEEE Conference on Decision and Control*, volume 3, pages 2709–2714, Phoenix, Arizona, USA, 1999.

[65] LBNL National Energy Research Scientific Computing Center(NERSC). Jacquard - Opteron Cluster. `http://www.nersc.gov/nusers/resources/jacquard/`, 2006.

[66] B. Nichols, D. Buttlar, and J. P. Farrell. *PThreads Programming*. A POSIX Standard for Better Multiprocessing. O'Reilly, Sebastopol, CA, First edition, 1996.

[67] NLANR/DAST. Iperf 1.7.0 : The TCP/UDP Bandwidth Measurement Tool. `http://dast.nlanr.net/Projects/Iperf/`, 2005.

[68] M. Parashar and J.C. Browne. Conceptual and Implementation Models for the Grid. *IEEE, Special Issue on Grid Computing*, 93(2005):653–668, 2005.

[69] M. Parashar, H. Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang, and S. Hariri. Automate: Enabling autonomic applications on the grid. *Cluster Computing*, 9(2):161–174, 2006.

[70] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using Control Theory to Achieve Service level Objectives in Performance Management. *Real Time Systems*, 23(1-2):127 – 141, 2002.

[71] J. S. Plank, M. Beck, W. R. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the Network. In *NetStore99*, Seattle, WA, USA, 1999.

[72] J.S. Plank, S. Atchley, Y. Ding, and M. Beck. Algorithms for High Performance, Wide-area Distributed File Downloads. *Parallel Processing Letters*, 13(2):207–223, 2003.

[73] J.S. Plank and M. Beck. The Logistical Computing Stack – A Design For Wide-Area, Scalable, Uninterruptible Computing. In *Dependable Systems and Networks, Workshop on Scalable, Uninterruptible Computing (DNS 2002)*, Bethesda, Maryland, USA, 2002.

[74] A. Rajasekar, M. Wan, and R. Moore. MySRB and SRB - components of a Data Grid. In *11th IEEE International High Performance Distributed Computing (HPDC-11)*, pages 301 – 310, Edinburgh, Scotland, 2002.

[75] A. Rowe, D. Kalaitzopoulos, M. Osmond, M. Ghanem, and Y. Guo. The Discovery Net System for High Throughput Bioinformatics. *Bioinformatics, Oxford University Press*, 19(1):i225–i231, 2003.

[76] K. Schwan, B. F. Cooper, G. Eisenhauer, A. Gavrilovska, M. Wolf, H. Abbasi, S. Agarwala, Z. Cai, V. Kumar, J. Lofstead, M. Mansour, B. Seshasayee, and P. Widener. *AutoFlow: Autonomic Information Flows for Critical Information Systems*, pages 275–303. CRC Press, New York, 2006.

[77] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS Management in Web Servers. In *Real-Time Systems Symposium*, pages 63–72, Cancun, Mexico, 2003.

[78] X. Shen, W. Liao, A. Choudhary, G. Memik, and M. Kandemir. A High-Performance Application Data Environment for Large-Scale Scientific Computations. *IEEE Transactions on Parallel and Distributed Systems*, 14(12):1262–1274, 2003. 1045-9219.

[79] A. Shoshani, A. Sim, and J. Gu. Storage Resource Managers: Middleware Components for Grid Storage. In *10th Goddard Conference on Mass Storage Systems and Technologies, 19th IEEE Symposium on Mass Storage Systems*, page 14, College Park, MD, USA, 2002.

[80] A. Sim, J. Gu, A. Shoshani, and V. Natarajan. Datamover: Robust Terabyte-Scale Multi-file Replication over Wide-Area Networks. In M. Hatzopoulos and Y. Manolopoulos, editors, *16th International Conference on Scientific and Statistical Database Management (SSDBM04)*, volume 00, pages 403–412, Santorini Island, Greece, 2004. IEEE Computer Society.

[81] R. Srikant. Control of communication networks. In T. Samad, editor, *Perspectives in Control Engineering: Technologies, Applications, New Directions*, pages 462–488. Wiley-IEEE Press, 2000.

[82] R. D. Stevens, A. J. Robinson, and C. A. Goble. myGrid: Personalised Bioinformatics on the Information Grid. In *11th International Conference on Intelligent Systems in Molecular Biology*, volume 19, pages i302–i304, Brisbane, Australia, 2003. Bioinformatics, Oxford University Press.

[83] I. Taylor, I. Wang, M. Shields, and S. Majithia. Distributed Computing with Triana on the Grid. *Concurrency and Computation: Practice & Experience*, 17(9):1197–1214, 2005.

[84] W. M. P. van-der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[85] S. Venugopal, R. Buyya, and L. Winton. A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids. *Concurrency and Computation: Practice & Experience*, 18(6):685 – 699, 2006.

[86] R. Vilalta, C. Apte, J. L. Hellerstein, S. Ma, and S. M. Weiss. Predictive Algorithms in the Management of Computer Systems. *IBM Systems Journal*, 41(3):461–474, 2002.

[87] VMware. Virtualization Basics. `http://www.vmware.com/virtualization/`, 2008.

[88] W3C. OWL Web Ontology Language Overview. `http://www.w3.org/TR/owl-features`, 2004.

[89] M. Wan, A. Rajasekar, R. Moore, and P. Andrews. A Simple Mass Storage System for the SRB Data Grid. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03)*, page 20, San Diego, California, USA, 2003. IEEE Computer Society.

[90] X. Wu and J. L. Hellerstein. Control Theory in Log Processing Systems. In *Summer 2005 RADS (Reliable Adaptive Distributed systems Laboratory) Retreat*, 2005.

Vita

Viraj Bhat

05/2008 PH.D. in Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ, USA

05/2003 M.S. in Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ, USA

06/1998 B.E. in Electrical Engineering, MANIT/REC, Bhopal, India

09/2004-04/2008 Graduate Assistant, CAIP, Rutgers University, NJ, USA

06/2007-09/2007 Software Engineer Intern, SSG-SPI (Software Solutions Group - Software Pathfinding Initiative) Intel, Santa Clara, CA, USA

01/2004-05/2007 Researcher, Princeton Plasma Physics Lab, Princeton, NJ, USA

05/2001-09/2002 Graduate Assistant, CAIP, Rutgers University, NJ, USA

07/1998-08/2000 Senior Software Engineer, Satyam Computer Services, Bangalore, India

Journal Publications

"**An Autonomic Data Streaming Service**," V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, S. Abdelwahed, and S. Klasky, *Cluster Computing: The Journal of Networks, Software Tools, and Applications, Special Issue on Autonomic Computing*, Springer Science+Business Media B.V. (Kluwer Academic Publishers), Hingham, MA, USA, Vol. 10, Issue 7, pp. 365-383, December 2007.

"**Autonomic Data Streaming for High Performance Scientific Applications**," V. Bhat, M. Parashar and N. Kandasamy, *Autonomic Computing: Concepts, Infrastructure and Applications*, Editors: M. Parashar and S. Hariri, CRC Press, pp. 413-433, 2006.

"**Data management on the fusion computational pipeline**," S. Klasky, M. Beck, V. Bhat, E. Feibush, B. Ludäscher, M. Parashar, A. Shoshani, D. Silver and M. Vouk, *Journal of Physics: Conference Series*, IOP Publishers, Vol. 16, pp. 510-520, 2005.

Conference Publications

"**Experiments with In-Transit Processing for Data Intensive Grid workflows**," V. Bhat, M. Parashar, and S. Klasky, Proceedings of the *8th IEEE International Conference on Grid Computing (Grid 2007)*, Austin, TX, USA, IEEE Computer Society, pp. 193-200, September 2007.

"**A Self-Managing Wide-Area Data Streaming Service using Model-based Online Control**," V. Bhat, M. Parashar, M. Khandekar, N. Kandasamy, and S. Klasky, Proceedings of the *7th IEEE International Conference on Grid Computing (Grid 2006)*, Barcelona, Spain, IEEE Computer Society Press, pp. 176-183, September 2006.

"**Enabling Self-Managing Applications using Model-based Online Control Strategies**," V. Bhat, M. Parashar, M. Khandekar, N. Kandasamy, and S. Abdelwahed, *Proceedings of the 3rd IEEE International Conference on Autonomic Computing (ICAC 2006)*, Dublin, Ireland, IEEE Computer Society Press, pp. 15-24, June 2006.

"**An Autonomic Service Architecture for Self-Managing Grid Applications**," H. Liu, V. Bhat, M. Parashar and S. Klasky, *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*, Seattle, WA, USA, IEEE Computer Society Press, pp. 132-139, November 2005.

"**High Performance Threaded Data Streaming for Large Scale Simulations**," V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, pp. 243-250, Pittsburgh, PA, USA, 2004.