A Self-Managing Wide-Area Data Streaming Service using Model-based Online Control

Viraj Bhat[#1], Manish Parashar[#2], Hua Liu[%3], Mohit Khandekar[*4], Nagarajan Kandasamy[*5], Sherif Abdelwahed[@6]
Scott Klasky[$7]

[#]*Department of Electrical and Computer Engineering, Rutgers University*
94 *Brett Road, Piscataway, NJ,* 08854*, USA*

[%]*Xerox Innovation Group, 800 Phillips Rd, Mailstop 128-30E Webster NY 14580*

Email:hua.liu@xeroxlabs.com

[1]virajb@caip.rutgers.edu, [2]parashar@caip.rutgers.edu

[*]*Department of Electrical and Computer Engineering, Drexel University*
3141 *Chestnut Street, Philadelphia, PA* 19104*, USA*

[3]mdk372@drexel.edu, [4]kandasamy@cbis.ece.drexel.edu

[@]*Institute for Software Integrated Systems, 2015*

[5]sherif@isis.vanderbilt.edu

Terrace Place Nashville, TN 37235, USA

[$]*Oak Ridge National Laboratory*
P.O. Box 2008*, Oak Ridge, TN,* 37831*, USA*

[6]klasky@ornl.gov

***Abstract*— Efficient and robust data streaming services are a critical requirement of emerging Grid applications, which are based on seamless interactions and coupling between geographically distributed application components. Furthermore the dynamism of Grid environments and applications requires that these services be able to continually manage and optimize their operation based on system state and application requirements. This paper presents a design and implementation of such a self-managing data-streaming service based on online control strategies. A Grid-based fusion workflow scenario is used to evaluate the service and demonstrate its feasibility and performance.**

I. Introduction

Grid computing has established itself as the dominant paradigm for wide-area high-performance distributed computing. As Grid technologies and testbeds mature, they are enabling a new generation of scientific and engineering application formulations based on seamless interactions and couplings between geographically distributed computational, data, and information services. A key requirement of these applications is the support for high-throughput low-latency robust data streaming between the corresponding distributed components. For example, a typical Grid-based fusion simulation workflow consists of coupled simulation codes running simultaneously on separate HPC resources at supercomputing centers. Further, they must interact at runtime with services for interactive data monitoring, online data analysis and visualization, data archiving, and collaboration that also run simultaneously on remote sites. The fusion codes generate large amounts of data, which must be streamed efficiently and effectively between these distributed components. Moreover, the data-streaming services themselves must have minimal impact on the execution of the simulations, satisfy stringent application/user space and time constraints, and guarantee that no data is lost.

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 Grid applications grow in scale and complexity, and with many of these applications running in batch mode with limited or no direct 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 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 Grid.

This paper presents the design, implementation, and experimental evaluation of such 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 [14]. A key contribution of this paper is the combination of typical rule-based self-management approaches with more formal model-based online control strategies. While the former are relatively simple and easy to implement, they require 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 paper combines model-based limited look-ahead controllers (LLC) with rule-based managers to dynamically achieve adaptive behaviour in Grid applications under various operating conditions [7].

This paper also illustrates and evaluates the operation of the data streaming service using a Grid-based fusion simulation workflow. This workflow consists 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 paper is organized as follows. Section II describes the driving Grid-based fusion simulation project and highlights its data streaming requirements and challenges. Section III describes the models and mechanism for enabling self-managing Grid services and applications. Section IV presents the design, implementation, operation and evaluation of the self-managing data streaming service. Section V presents the evaluation of rule driven autonomic services and the model based online control in conjunction with the rule driven services. Section VI addresses the scalability of the service and then proposes as well as evaluates hierarchical control strategies. Section VII presents related work. Section VIII concludes the paper.

II. Wide-Area Data Streaming in the Fusion Simulation Project

A. Fusion Simulation Workflow

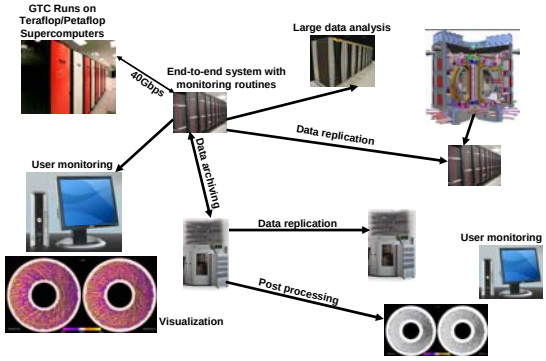

Fig. 1 A workflow for the Fusion Simulation Project

The DoE SciDAC CPES fusion simulation project [11] is developing 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 simulation data are critical to this project and the scientific discovery process. Fig. 1 shows a typical workflow comprising of coupled simulation codes – the edge turbulence particle-in-cell (PIC) code (GTC) and the microscopic MHD code (M3D) – running simultaneously on thousands of processors at various supercomputing centers. The 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.

B. Requirements for a wide-area data streaming service

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) Minimize related overhead on the executing simulation. Since the simulation is long running and executes in batch for days, the overhead due to data streaming on the simulation should be less than 10% of the simulation execution time. (3) Adapt to network conditions to maintain desired QoS. The network is a shared resource and the usage patterns vary constantly. (4) Handle network failures while eliminating data loss. Network failures can lead to buffer overflows, and data has to be written to local disks to avoid loss. However, this increases overhead on the simulation and the data is not available for real-time remote analysis and visualization.

III. Model, Mechanisms and Infrastructure for Self-Management

The data streaming service described in this paper is constructed using the Accord programming infrastructure [14], which provides the core models and mechanisms for realizing self-managing Grid services. These include models and mechanisms for autonomic management using rules as well as model-based online control. Its key components described in the following sections.

A. The Accord Autonomic Services Architecture

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

The Accord autonomic service architecture extends the service-based Grid programming paradigm to relax assumptions of static (defined at the time of instantiation) application requirements and system/application behaviours, and allow them to be dynamically specified. It also enables the behaviours 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 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 behaviours based on current requirements, state and execution context.

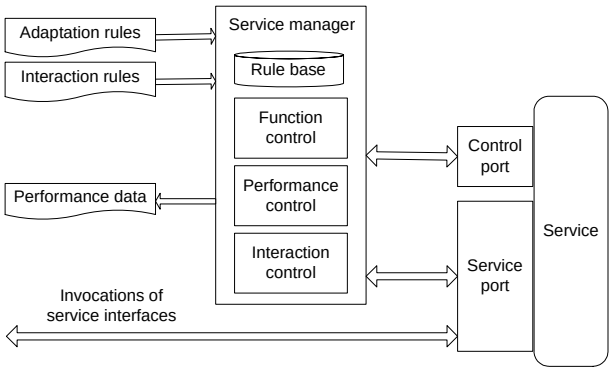

Fig. 2 An autonomic service in Accord

1) Definition of Autonomic Services

An autonomic service (see Fig. 2) 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[9] 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 Fig. 8. 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 Fig. 9.

2) The Runtime Infrastructure

The Accord runtime infrastructure (shown in Fig. 3) consists of a user/developer portal, peer service and application composition/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.

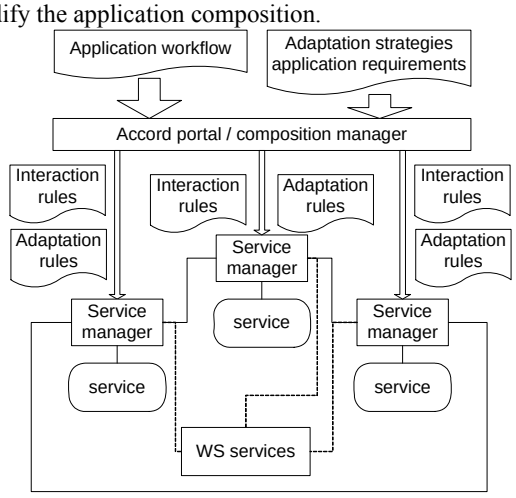

Fig. 3 Accord runtime infrastructure: Solid lines indicate interactions among services and dotted lines represent invocation of WS instances providing supporting services such as naming and discovery.

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 [13].

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.

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 [13]. The service organization describes a collection of services that are functionally compose-able, determined semantically (e.g., using OWL [27]) or syntactically using WSDL [9]. 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 [4]), 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 [26] in the workflow into corresponding rule templates [13]. 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/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 [13].

B. Model-Based Control within Accord

Fig. 4 shows the overall framework of a *limited look-ahead controller* (*LLC*) [1] where the QoS management problem is posed as one of sequential optimization under uncertainty. Relevant operating parameters of the Grid environment such as data-generation patterns and network bandwidth are estimated and used by a mathematical model to forecast future application behaviour over a prediction horizon N. The controller optimizes the forecast behaviour as per the specified QoS goals by selecting the best control inputs to apply to the system. At each time step k, the controller finds a feasible sequence $\{u^*(i) \mid i \in [k+1, k+N]\}$ of control 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.

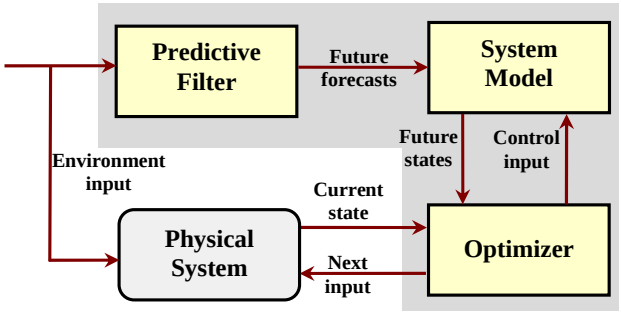

Fig. 4 The LLC control structure

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 that exhibit non-linear behaviour and where control or tuning inputs must be chosen from a finite set.

As shown in Fig. 5, the element (service) managers within the Accord programming system are augmented with online controllers [7]. Each manager monitors the state of its underlying elements and their execution context, collects and reports runtime information, and enforces the adaptation actions decided by the controller. These 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 behaviour and the managers focus on enforcing these adaptations in a consistent and efficient manner.

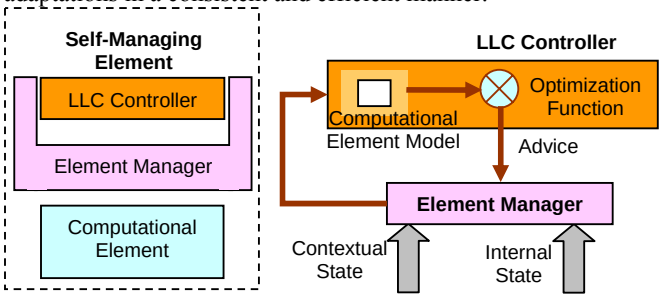

Fig. 5 A self-managing element and the interaction between an element manager and the corresponding controller

IV. The Self-Managing Data-Streaming Service

This section describes a self-managing data streaming services for the Grid-based fusion simulation workflow based on the models and mechanisms presented in the previous section. A specific driving simulation workflow is shown in

Fig. 6, 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 analysed 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).

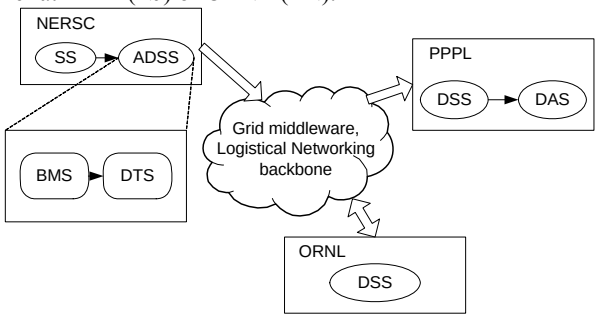

Fig. 6 A self-managing data streaming service

The data streaming service in Fig. 6 has four core services: (1) A *Simulation Service* (SS) executing on an IBM SP machine at NERSC, and generating data at regular intervals; (2) A *Data Analysis Service* (DAS) executing on a computer cluster located at PPPL to analyse the data streamed from NERSC; (3) A *Data Storage Service* (DSS) to archive the streamed data using the Logistical Networking backbone [21], 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).

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*: We would like 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.

A. Design of the ADSS Controller

The ADSS controller is designed using the LLC concepts discussed in Section III. Fig. 7 shows the system 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:

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

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

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 (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 $0 \le \mu_i(k) \le 1$ and $0 \le \omega_i(k) \le 1$ 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 (as discussed in Section IV-B).

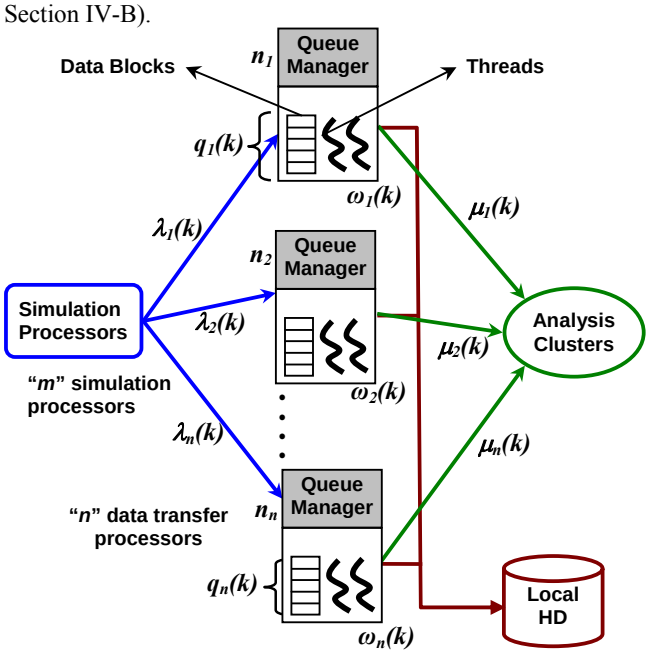

Fig. 7 The system model for the data streaming service

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

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

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

Here, N_p denotes the prediction or look-ahead horizon, q_{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. Since the execution time available to the controller is often limited by hard application bounds, it is necessary to consider the possibility that we may have to deal with suboptimal solutions. For adaptation purposes, however, it is not critical to find the global optimum to ensure system stability; a feasible suboptimal solution will suffice. However, we would still like to use the available time exploring the most promising solutions leading to optimality. Taking advantage of the fact that the operating environment does not change drastically over a short period of time, we can obtain suboptimal solutions using *local search methods*, where given the current values of $\mu_i(k)$ and $\omega_i(k)$, the controller searches a limited neighbourhood of these values for a feasible solution for the next step.

B. Implementation and Deployment of ADSS

ADSS is a composite service comprising a *Buffer Manager Service* (BMS) managing the buffers allocated by the ADSS, and a *Data Transfer Service* (DTS) managing the transfer of data blocks from the buffers to remote services for analysis and visualization at PPPL, and archiving at PPPL or ORNL. The BMS supports two buffer management schemes. *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 is used when the network is congested. The control ports for these services are described in [14].

A EWMA filter with a smoothing constant of 0.5 estimates the data generated by the simulation for the *ADSS controller*. A single-step LLC strategy is used with a desired buffer size of $q^* = 0$ on each node n_i. 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 our implementation.

The *ADSS Element Manager* supplies the controller with internal state of the ADSS and SS services, including the observed buffer size on 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 [20], which reports the bandwidth available to datagram packets in the TCP protocol, and their delay jitter and loss rate. The element manager also stores a set of rules which are triggered based on controller decisions.

The element manager triggers adaptations within the DTS/BMS service. 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, e.g., NERSC/ORNL, when the network is congested.

V. Evaluation of the Self-Managing Data-Streaming Service

This section presents an evaluation of the Accord-based self-managing data-streaming service. The first group of experiments evaluate the rule-based adaptations while the second group evaluates a combination of rule-based and control-based adaptations. A comparison of the two strategies and the overhead of the self managing data streaming service are also presented.

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 (peak) ESNET connection between PPPL and NERSC was used. A single controller was used, and the controller and managers were implemented using threading. A maximum of four simulation processors were used for data streaming.

A. 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 Fig. 8. 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 Fig. 9, 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.

```
<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>
```

Fig. 8 The control port for the BMS

The resulting adaptation behaviour is plotted in Fig. 10(a). The figure shows that BMS switches to aggregate buffer management during simulation time intervals 75 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. Fig. 10 (b) 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.

```
<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>
```

Fig. 9 The adaptation rule for BMS

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.

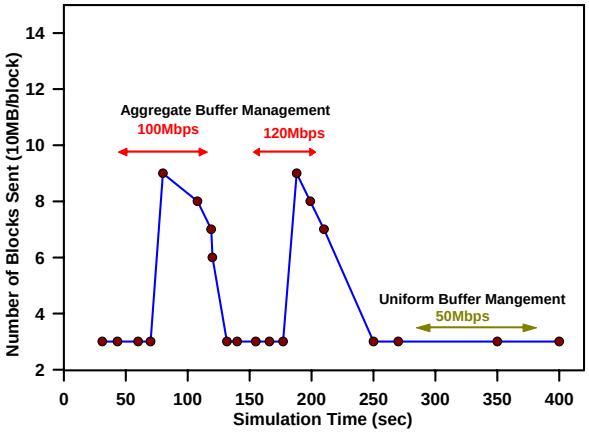

Fig. 10(a) Self-optimization behaviours of the Buffer Management Service (BMS) – BMS switches between uniform blocking and aggregate blocking algorithms based on application data generation rates, network transfer rates and the nature of data generated

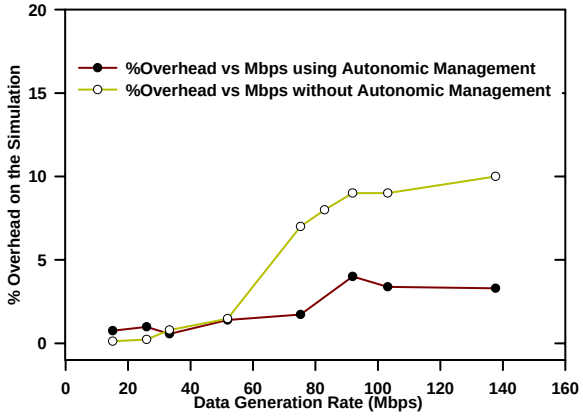

Fig. 10(b) Percentage overhead on simulation execution with and without autonomic management using rules

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

The effectiveness of the data transfer between the simulation service at NERSC and the analysis/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 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 [6].

```
<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>
```

Fig. 11 The adaptation rule for the ADSS

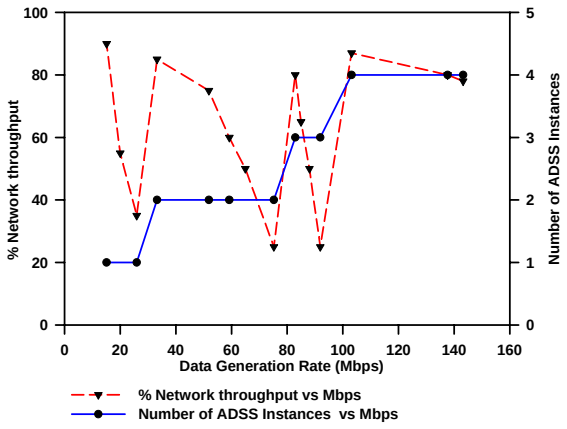

Fig. 12 Effect of creating new instances of the ADSS service when the %Network Throughput dips to below the user defined 50% threshold

The self-configuration behaviour of ADSS is governed by the rule shown in Fig. 11. When the network transfer rate is below a pre-defined threshold, ADSS will use Accord to create new instances of ADSS including BMS and 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 Fig. 12. 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 Fig. 11. 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 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>
<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>
</rule>
```

Fig. 13 The interaction/adaptation rule for 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 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 [6]. 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 [12] 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 Fig. 13.

The resulting self-healing behaviour is plotted in Fig. 14. 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.

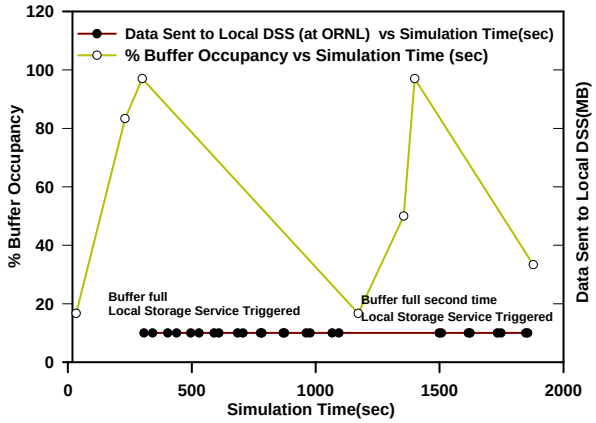

Fig. 14 Effect of switching from the DSS at PPPL to the DSS ORNL in response to network congestion and/or failure

B. Self-Managing Scenarios using Rule and Control based Adaptations

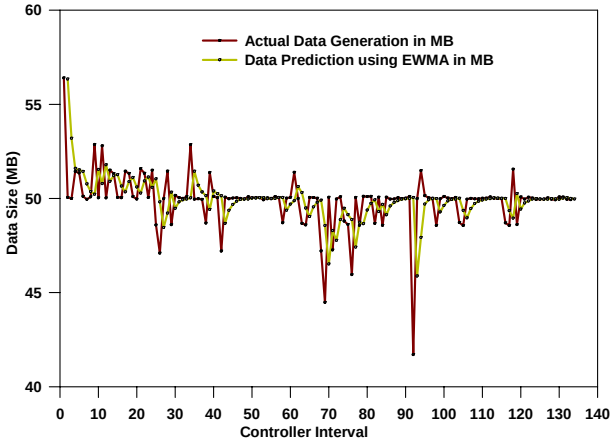

Fig. 15 Actual and predicted data generation rates for the GTC simulation

Predicting data generation rates: Fig. 15 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: $\hat{\lambda}_i(k) = \gamma \cdot \lambda_i(k) + (1-\gamma) \cdot \hat{\lambda}_i(k-1)$ where γ = 0.5 is the smoothing factor.

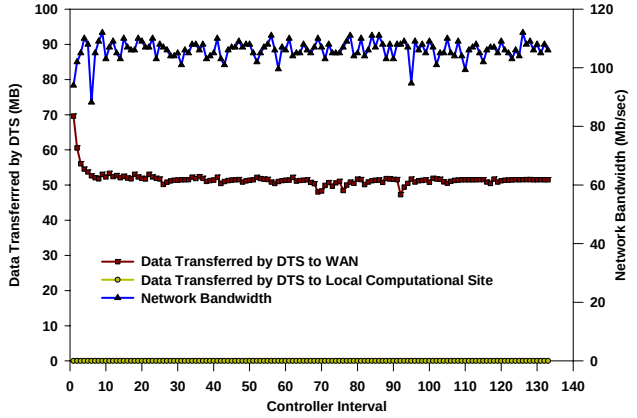

Fig. 16 Controller and DTS operation for the GTC simulation

Controller behaviour for long-running simulations: Fig. 16 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.

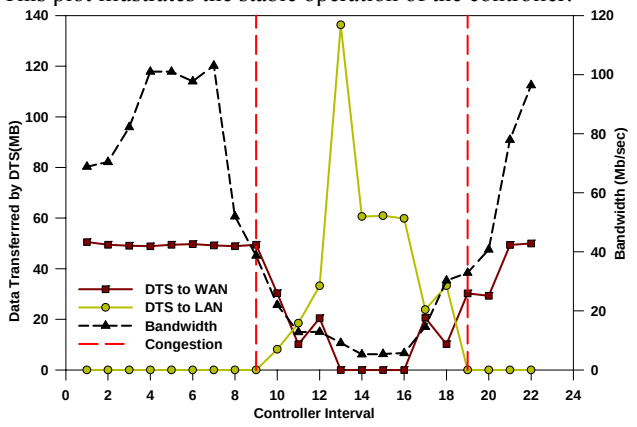

Fig. 17 DTS adaptation due to network congestion

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 Fig. 8. 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 Fig. 17 that this adaptation is triggered multiple times until the network is no longer congested at around the 19th controller interval. The data sent to the local storage falls to zero at this point.

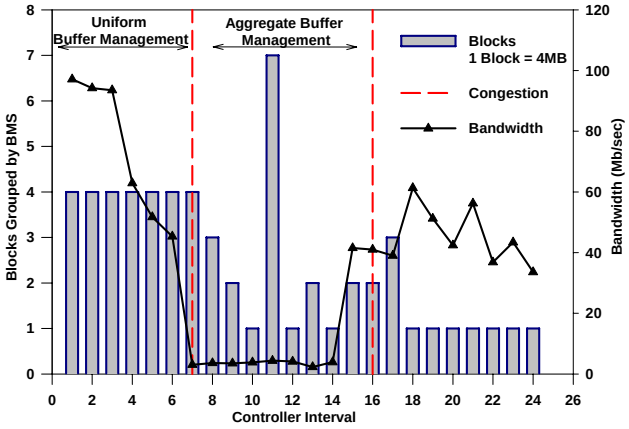

Fig. 18 BMS adaptations due to varying network conditions

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. Fig. 18 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.

C. Comparison of Rule-based and Control-based Adaptation in the ADSS

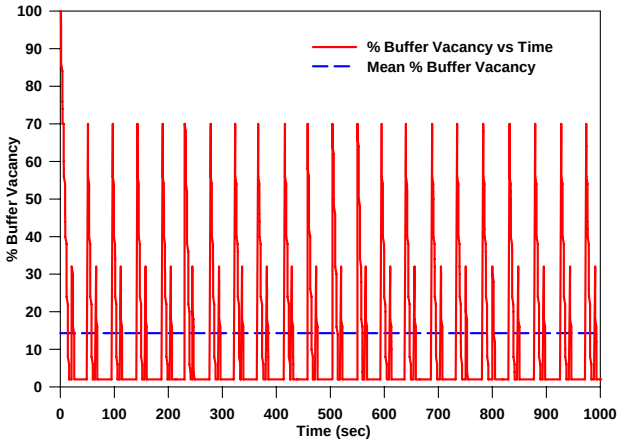

Fig. 19 % Buffer Vacancy using heuristically based rules

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. Fig. 19 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.

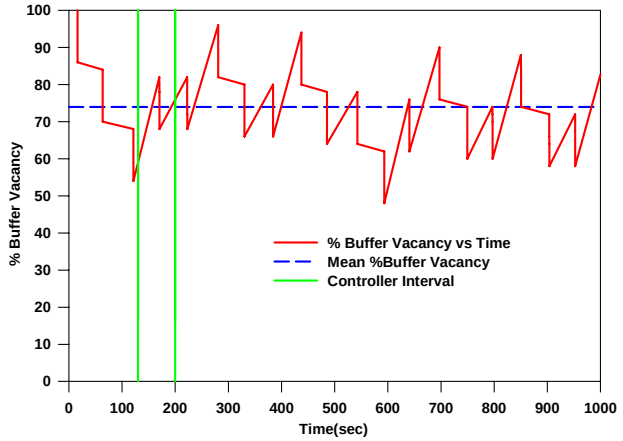

Fig. 20 % Buffer Vacancy using control-based self-management

Such a high occupancy leads to a slow down of the simulation [6] and also results in increased loss of data due to buffer overflows. Fig. 20 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.

D. 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 [19]) 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: $(T^*_s - T_s)/T_s$, where 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.

VI. 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 Fig. 21, 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.

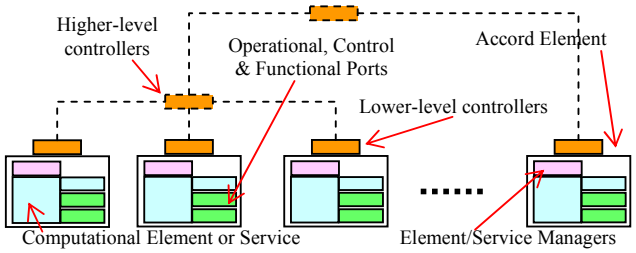

Fig. 21 Constructing a hierarchy of controllers in Accord

The Accord runtime framework ensures coordinated and 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.

A. Hierarchical Control for Data Streaming

Recall that when control inputs must be chosen from a set of discrete values, the optimization problem described in Section IV-A 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 Fig. 22. Here, we have a larger system compared to the one described in Section IV-B – 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 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. Fig. 22 shows L1 and L0 controllers within a two-level hierarchy working together to achieve the desired QoS goals with the following responsibilities.

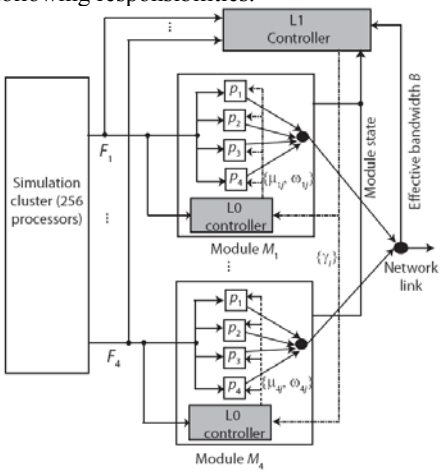

Fig. 22 Hierarchical Controller Formulation for Data Streaming

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 $\{\gamma_i\}$, 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 IV. 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.

The hierarchical structure in Fig. 22 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 Fig. 22, each L1 controller must know the approximate behaviour of the components comprising the L0 level. For example, to solve the combinatorial optimization problem of determining $\{\gamma_i\}$, the fraction of the available network bandwidth to allocate to the modules, the L1 controller must be able to quickly approximate the behaviour 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 behaviour 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 [5] to generate a look-up table that quickly approximates M_i's behaviour. Here, M_i's behaviour 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 quickly generate decisions for use in real time.

B. Simulation Results

Fig. 23 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 $0 \le \gamma_i \le 1$ supplied by the L1 controller to each M_i was quantized in intervals of 0.1.

Fig. 24 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.

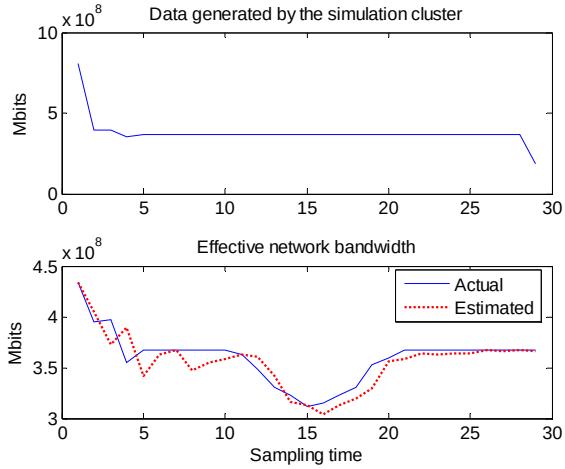

Fig. 23 GTC workload trace and effective network bandwidth between NERSC and PPPL

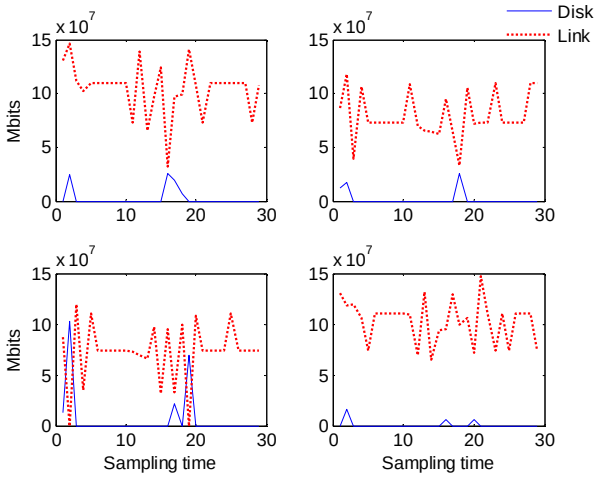

Fig. 24 Operation of the L0 and L1 controllers

VII. RELATED WORK

There have been several recent research efforts addressing self-management. Some have investigated runtime adaptations by tuning TCP buffers (*Enable* [25] and GridFTP [24]). Others use feedback (or reactive) control for resource and performance management for single-processor application [10], application task scheduling [8], [16], bandwidth allocation and QoS adaptation in web servers [3], load balancing in e-mail and file servers [15], network flow control [18], [23] and processor power management [17], [22]. 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 this paper is to address this limitation and manage the performance of Grid applications, which exhibit hybrid behaviour comprising both discrete-event and time-based dynamics [2], and execute under explicit operating constraints, using the LLC method. Predictive and change-point detection algorithms have been proposed for managing application performance.

VIII. CONCLUSION

The paper 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 the application 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 this scheme.

ACKNOWLEDGMENT

The research presented in this paper is supported in part by National Science Foundation via grants numbers ACI 9984357, EIA 0103674, EIA 0120934, ANI 0335244, CNS 0305495, CNS 0426354, IIS 0430826, and by Department of Energy via the grant number DE-FG02-06ER54857.

REFERENCES

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

[2] S. Abdelwahed, G. Karsai and G. Biswas, *Online Safety Control of a Class of Hybrid Systems*, *IEEE 2002 Conference on Decision and Control*, Las Vegas, NV, 2002, pp. 1988-1990.

[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 (2002), pp. 80-96.

[4] M. Agarwal and M. Parashar, *Enabling Autonomic Compositions in Grid Environments*, *Fourth International Workshop on Grid Computing (Grid '03)*, IEEE Computer Society, Phoenix, Arizona, USA, 2003, pp. 34-41.

[5] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, Nashua, NH, USA, 2005.

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

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

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

[9] 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

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

[11] 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), pp. 510-520.

[12] Lawrence-Berkeley-National-Laboratory, *Energy Sciences Network*, http://www.es.net/, 2004

[13] H. Liu, *Accord: A Programming System for Autonomic Self-Managing Applications*, *Electrical and Computer Engineering*, Rutgers University, Piscataway, NJ, USA, 2005, pp. 104.

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

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

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

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

[18] S. Mascolo, *Classical Control Theory for Congestion Avoidance in High-Speed Internet*, *38th IEEE Conference on Decision and Control*, Phoenix, Arizona, USA, 1999, pp. 2709-2714.

[19] B. Nichols, D. Buttlar and J. P. Farrell, *PThreads Programming*, O'Reilly, Sebastopol, CA, 1996.

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

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

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

[23] R. Srikant, *Control of Communication Networks*, in T. Samad, ed., *Perspectives in Control Engineering: Technologies, Applications, New Directions*, Wiley-IEEE Press, 2000, pp. 462-488.

[24] S. Thulasidasan, W. Feng and M. K. Gardner, *Optimizing GridFTP through Dynamic Right-Sizing*, *12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Seattle, WA, USA, 2003, pp. 14-23.

[25] B. L. Tierney, D. Gunter, J. Lee, M. Stoufer and J. B. Evans, *Enabling Network-Aware Applications*, *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, San Francisco, CA, USA, 2001, pp. 281-288.

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

[27] W3C, *OWL Web Ontology Language Overview*, http://www.w3.org/TR/owl-features, 10 February 2004