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

Viraj Bhat[#1], Manish Parashar[#2], Mohit Khandekar[*3], Nagarajan Kandasamy[*4], Scott Klasky[$5]

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

[$]*Oak Ridge National Laboratory*
P.O. Box 2008*, Oak Ridge, TN,* 37831*, USA*
[5]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 quality-of-service (QoS) 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, and 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 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 [10]. A key contribution of this paper 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 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 [6].

This paper 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 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 addresses the scalability of the service and proposes and evaluates hierarchical control strategies. Section VI presents related work. Section VII concludes the paper.

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

A. Fusion Simulation Workflow

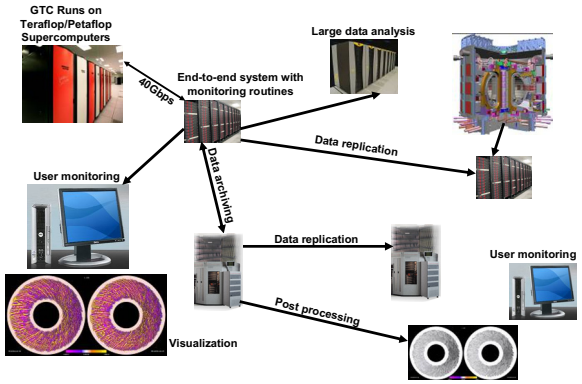

Fig. 1 A workflow for the Fusion Simulation Project

The DoE SciDAC CPES fusion simulation project [9] 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. The simulation executes in batch for days and we would like the overhead due to data streaming on the simulation to 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 typically vary constantly; and (4) Handle network failures while eliminating data loss. Network failures usually 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 and Mechanisms for Self-Management

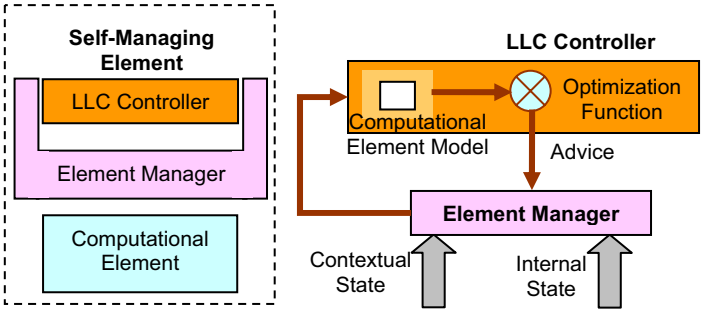

Fig. 2 A self-managing element and the interaction between an element manager and the corresponding controller

The data streaming service presented in this paper is constructed using the Accord infrastructure [10], which provides the core models and mechanisms for realizing self-managing Grid services. Its key components, shown in Fig. 2, are described in the following sections.

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 behaviour and allow them to be dynamically specified using high-level rules. It also enables the behaviour 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 self-managing functional, interaction, and composition behaviour 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 behaviour 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 (e.g., parameters, variables, or functions) that can query the operating state of the service and modify its behaviour. It is described using WSDL (Web Service Definition Language) 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 port and policy specifications can be found in [10].

B. Model-Based Control

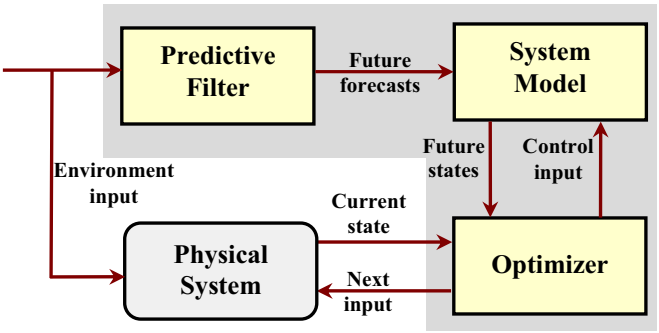

Fig. 3 The LLC control structure

Fig. 3 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.

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.

The element (service) managers within the Accord programming system are augmented with online controllers [6]. 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.

IV. 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 section. A specific driving simulation workflow is shown in Fig. 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 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).

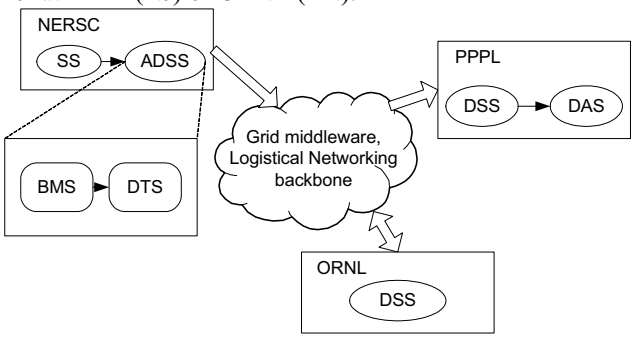

Fig. 4 A self-managing data streaming service

The data streaming service in Fig. 4 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 [16], 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. 5 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, 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 $0 \le \mu_i(k) \le 1$ 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 (discussed in Section IV-B).

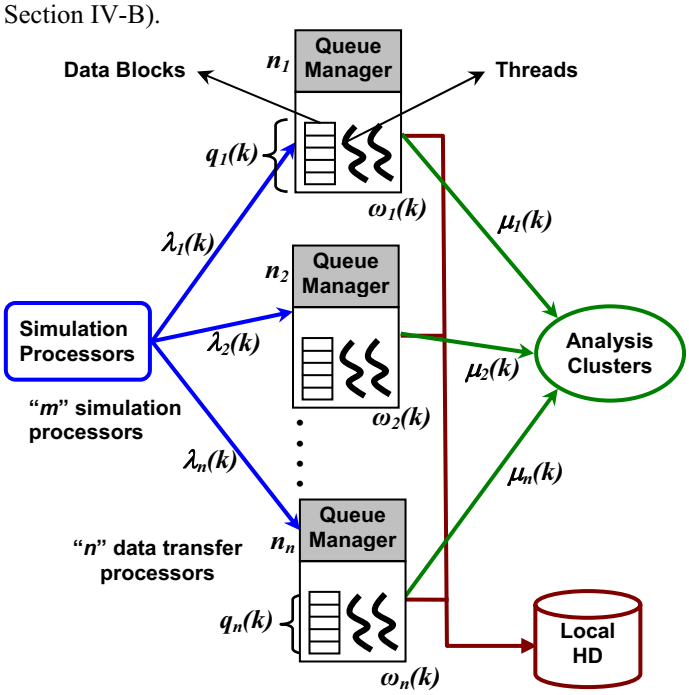

Fig. 5 The system model for the data streaming service

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.

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

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

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 for 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 [10].

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 [15], 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.

C. 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.

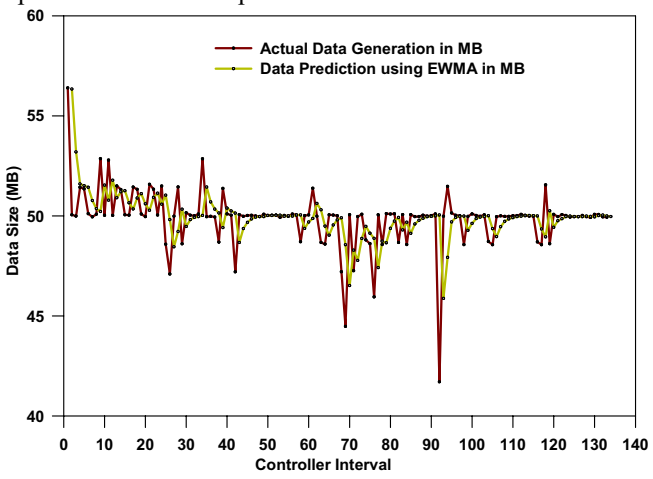

Fig. 6 Actual and predicted data generation rates for the GTC simulation

Predicting data generation rates: Fig. 6 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.

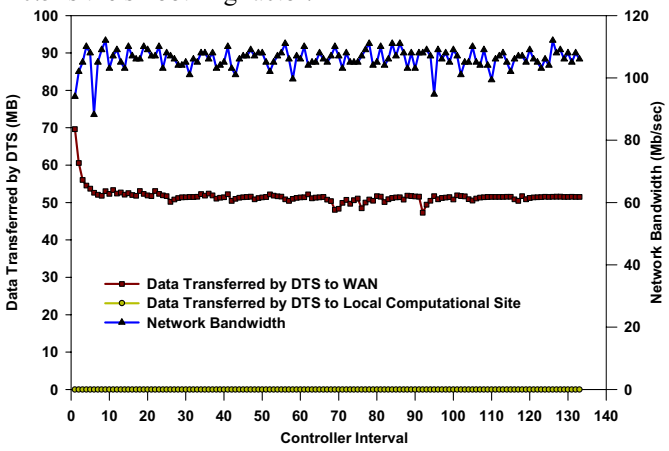

Fig. 7 Controller and DTS operation for the GTC simulation

Controller behaviour for long-running simulations: Fig. 7 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.

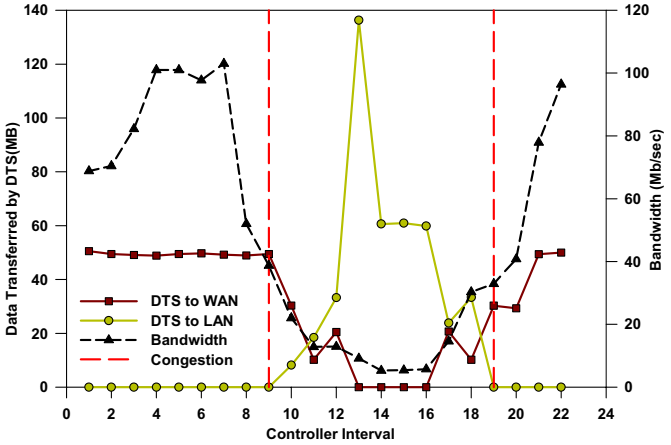

Fig. 8 DTS adaptation due to network congestion

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. 8 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.

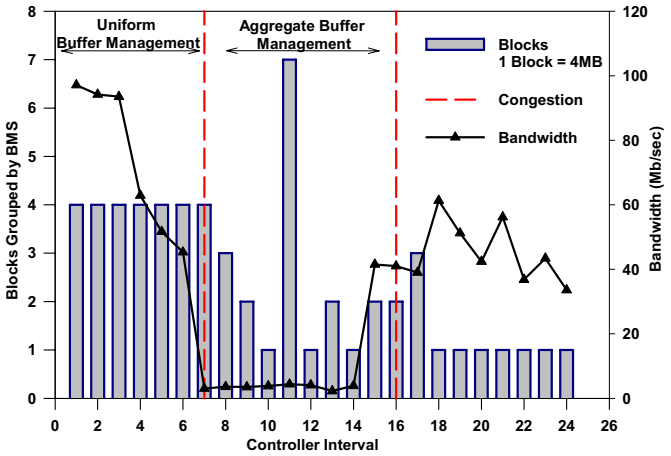

Fig. 9 BMS adaptations due to varying network conditions

***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. 9 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.

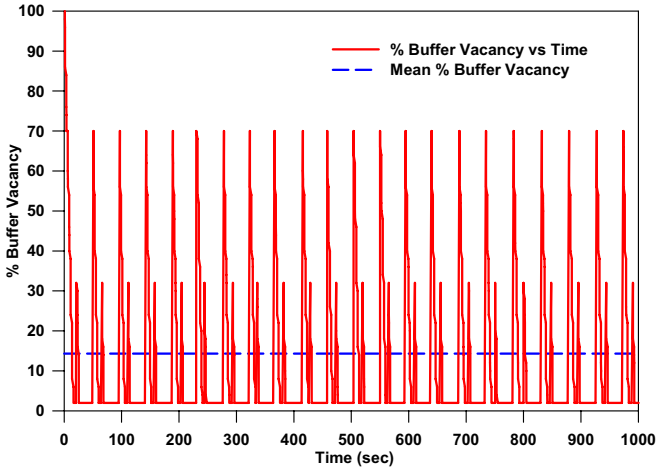

Fig. 10 % Buffer Vacancy using heuristically based rules

Heuristic vs. Control Based Adaptation in 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. Fig. 10 plots the % buffer vacancy for the first case. In this case, management was purely reactive and based on heuristics. 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.

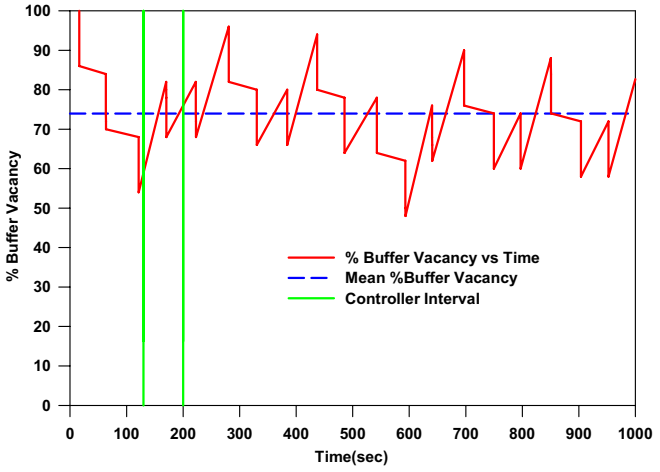

Fig. 11 % Buffer Vacancy using control-based self-management

Such a high occupancy leads to a slow down of the simulation [5] and also results in increased loss of data due to buffer overflows. Fig. 11 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.

Controller Overhead: 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.

V. Addressing Scalability Using Hierarchical Control

As the system scale increases, the centralized solution presented in the previous section will not be feasible for real-time controller operation. To address this issue we explore the hierarchical control architecture in this section. Simulations are used to demonstrate its effectiveness.

A. System Model and Control Structure

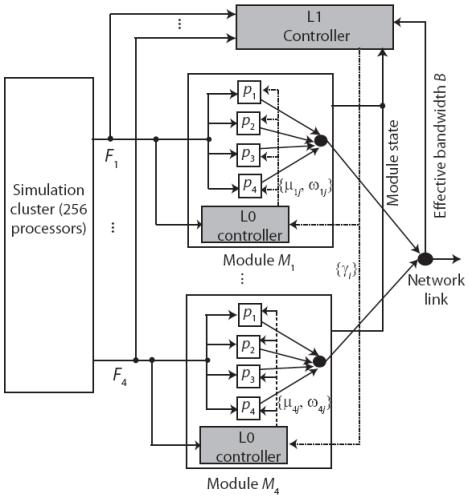

Fig. 12 Hierarchical Controller Formulation for Data Streaming

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. 12. Here, we have a larger system compared to the one described in Section IV-C – 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. 12 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 $\{\gamma_i\}$, 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. 12 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. 12, 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 [4] 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

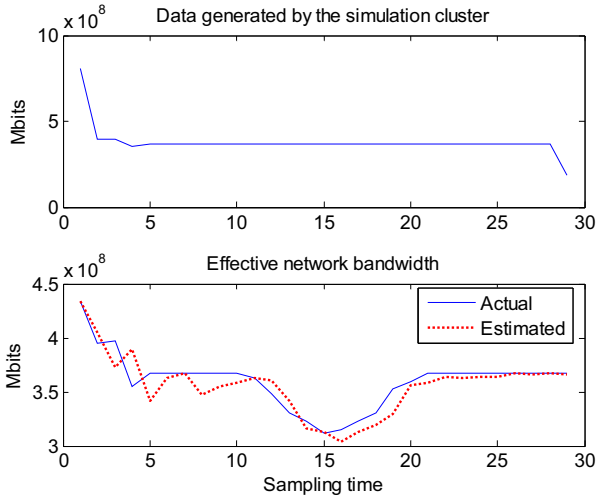

Fig. 13 GTC workload trace and effective network bandwidth between NERSC and PPPL

Fig. 13 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.

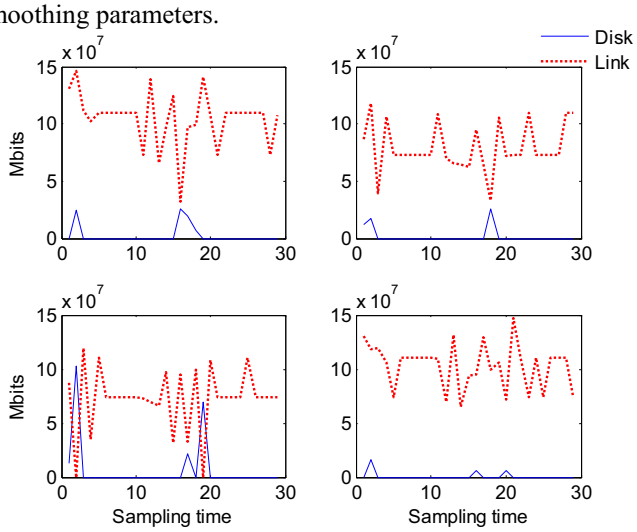

Fig. 14 Operation of the L0 and L1 controllers

Fig. 14 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.

Fig. 14 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.

VI. RELATED WORK

There have been several recent research efforts addressing self-management. Some have investigated runtime adaptations by tuning TCP buffers (*Enable* [20] and GridFTP [19]). Others use feedback (or reactive) control for resource and performance management for single-processor application [8], application task scheduling [7], [12], bandwidth allocation and QoS adaptation in web servers [3], load balancing in e-mail and file servers [11], network flow control [14], [18] and processor power management [13], [17]. 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.

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

[5] 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, USA, 2004, pp. 243-250.

[6] 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 (ICAC'06)*, Dublin, Ireland, 2006, pp. 15-24.

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

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

[9] 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.

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

[11] 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, USA, 2002, pp. 219-230.

[12] 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.

[13] 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)*, Grenoble, France, 2002, pp. 156-163.

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

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

[16] 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, USA, 2002.

[17] 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.

[18] 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.

[19] 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.

[20] 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.