Experiments with In-Transit Processing for Data Intensive Grid Workflows

Viraj Bhat [#1], Manish Parashar [#2], Scott Klasky [*3]

[#] *Department of Electrical and Computer Engineering, Rutgers University*
94 Brett Road, Piscataway, NJ 08854-8058, U.S.A.
[1] virajb@caip.rutgers.edu
[2] parashar@caip.rutgers.edu

[*] *Oak Ridge National Laboratory*
P.O. Box 2008, Oak Ridge, TN, 37831, USA
[3] klasky@ornl.gov

***Abstract*— Efficient and robust data streaming and in-transit data manipulations are critical requirements of emerging scientific and engineering application workflows, which are based on seamless interactions and coupling between geographically distributed application components. The overall goal of this research is to address these requirements and develop a data streaming and in-transit data manipulation service. In this paper, we experimentally investigate reactive management strategies for in-transit data manipulation, as well as cooperative end-to-end management for wide-area data-streaming and in-transit data manipulation for data-intensive scientific and engineering workflows.**

I. Introduction

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

The overall goal of this research to develop a data streaming and in-transit data manipulations service that provides the mechanisms as well as the management strategies for data intensive scientific and engineering workflows to addresses the requirements outlined above. In our previous work we addressed efficient and robust wide-area data streaming, and developed autonomic management strategies based on online control [1]. The developed service minimizes overheads on the simulations, provided proactive model-based Quality of Service (QoS) control at the data source, and avoids loss of data.

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

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

presents related work. Section VI concludes the paper and presents future work.

II. The Fusion Simulation Project and its Data Streaming Requirements

A. Fusion Simulation Workflow

The overarching goal of the DOE SciDAC CPES fusion simulation project [2] is to develop a new integrated Grid-based predictive plasma edge simulation capability to support next-generation burning plasma experiments, such as the International Thermonuclear Experimental Reactor (ITER). Effective online management and transfer of the simulation data is a critical part for this project and is essential to the scientific discovery process. It consists of coupled simulation codes, i.e., the edge turbulence particle-in-cell (PIC) code XGC coupled with Nimrod and the microscopic MHD code (M3D), which run simultaneously on thousands of processors on separate HPC resources at possibly distributed supercomputing centers. The data produced by these simulations must be streamed at runtime, to remote sites, for online simulation monitoring and control, simulation coupling, data analysis and visualization, online validation, and archiving. Furthermore, the data may have to be processed enroute to the destination as the data may have to transformed to match the coordinate system, representation, format, distribution and mapping, etc., of the destination node. Similarly, features of interest may need to be extracted and processed enroute to visualization or monitoring applications.

B. Data Streaming and In-Transit Processing Requirements

The fundamental requirement for a wide area data streaming and in-transit data processing service is to efficiently and robustly stream data from live simulations to remote services while satisfying the following constraints: (1) Enable high-throughput, low-latency data transfer to support near real-time access to the data. (2) Minimizing overheads on the executing simulation. The simulation executes in batch for days and we would like the overhead of the streaming on the simulation to be less than 10% of the simulation execution time. (3) Adapting to network conditions to maintain desired QoS. The network is a shared resource and the usage patterns typically vary constantly. (4) Handle network failures while eliminating loss of data. Network failures usually lead to buffer overflows, and data has to be written to local disks to avoid loss. This increases the overhead in the simulation. Further, the data is no longer available for remote analysis. (5) Effectively schedule and manage in-transit processing while satisfying the above requirements - this is particularly challenging due to the limited capabilities and resources and the dynamic capacities of the typically shared processing nodes.

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

A conceptual overview of the self-managing data streaming and in-transit processing service for Grid-based data intensive scientific workflows is presented in Figure 1. It consists of

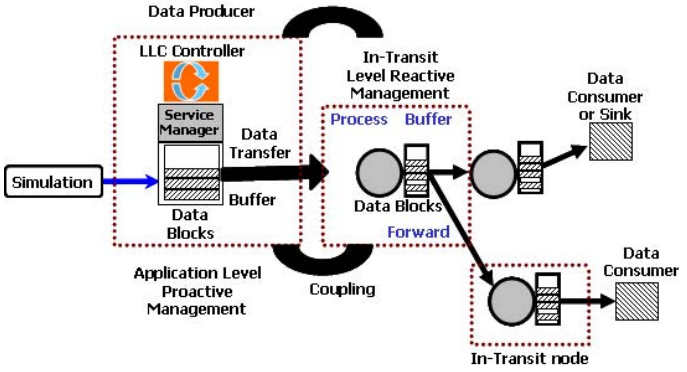

Fig. 1. Conceptual overview of the self-managing data streaming and in-transit processing service.

two key components: The first is an application level data streaming service, which provides adaptive buffer management mechanisms and proactive QoS management strategies based on online control and user-defined polices, at application end-points. The second component provides scheduling mechanisms and adaptive runtime management strategies for in-transit data manipulation and transformation. These two components work cooperatively to address the overall application constraints and QoS requirements outlined in Section II-B. The first component has been addressed in our previous work [1] and is briefly summarized below. This paper focuses on the second component and experiments with different in network processing strategies as well as their couplings with the application level mechanisms.

A. Application Level Data Streaming

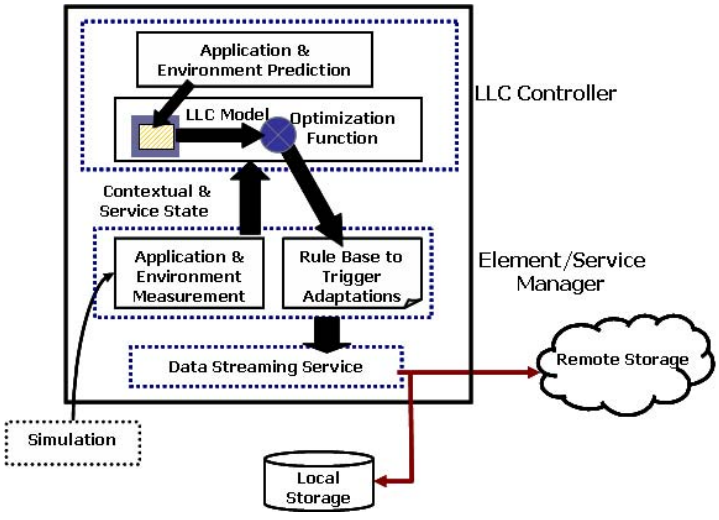

Fig. 2. A self-managing application level data streaming service.

The application level self-managing data streaming service combines model-based limited look-ahead controllers (LLC) and rule-based autonomic managers with adaptive multi-threaded buffer management and data transport mechanisms at

the application endpoints. It is constructed using the Accord-WS infrastructure for self-managing Grid services [3] and supports high throughput, low latency, robust application level data streaming in wide-area Grid environments as demonstrated in [1]. The autonomic data streaming service is illustrated in Figure 2 and consists of a service manager and an LLC controller. The service manager monitors the state of the service and its execution context, collects and reports runtime information, and enforces the adaptation actions determined by its controller. Augmenting the element manager with an LLC controller allows human defined adaptation polices, which may be error-prone and incomplete, with mathematically sound models and optimization techniques for more robust self-management. Specifically, the controller decides when and how to adapt the application behavior and the service managers focus on enforcing these adaptations in a consistent and efficient manner. The structure of the LLC-based online controller

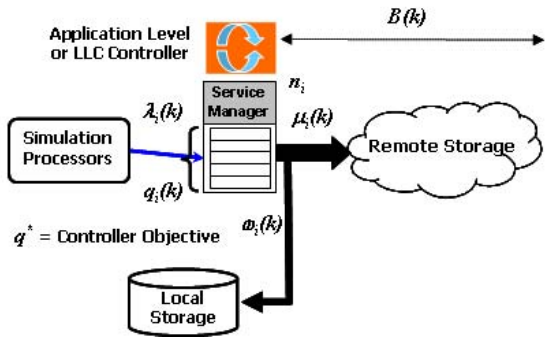

Fig. 3. Design of the LLC controller for an application level data streaming service.

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

The self-managing service behaves as follows. The element manager supplies the LLC controller (as shown in Figure 2) with information about the internal state of the application and the environment, which includes the observed buffer size, simulation-data generation rate, and the network bandwidth. When the controller detects congestion due to a decrease in parameter $B(k)$, it advises the service manager to increase $\omega_i(k)$ and decrease $\mu_i(k)$ to avoid loss of simulation data. The element manager contains the set of rules, which are invoked based on the controller's advice or decisions to adapt the service. For example, the controller decides the amount of data to be sent over the network or to local storage, and the service manager uses the controllers advise to select the corresponding buffer management scheme to be used within the data streaming service to achieve this. The element manager can also adapt the data streaming service to send data to local storage rather than streaming it to a remote site when the network is congested. Experimental evaluation of the application level data streaming service in a wide area Grid environment demonstrate its scalability, stability, its ability to effectively maintain application QoS and avoid data loss, as well as its low overheads on the simulation [1].

B. In-Transit Data Manipulations

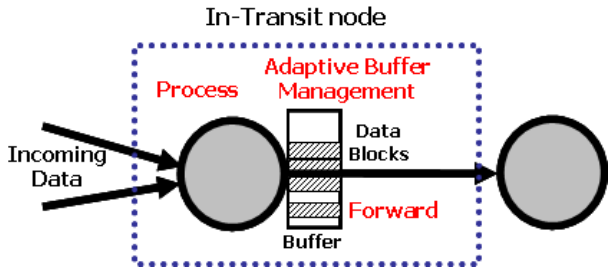

Fig. 4. Architecture of an in-transit node.

The in-transit data manipulation framework consists of a dynamic overlay of available in-transit processing nodes (e.g., workstations or small to medium clusters) with heterogeneous capabilities and loads. Note that these nodes may be shared across multiple workflows. The conceptual architecture of a node is illustrated in Figure 4. Each node performs three steps, viz., processing, buffering and forwarding. The processing depends on the capacity and capability of the node and the amount of processing that is still required. The basic idea is that each node completes at least its share of the processing (which may be predetermined or dynamically computed) and can perform additional processing if the network is too congested for forwarding. The amount of processing completed is logged in the data block itself. The goal of the in-transit processing is to process as much data as possible before the data reaches the sink. A processing that is not completed in-transit will have to be performed at the sink. The current design of the framework assumes that each node can perform any of the required data manipulations functions. Each in-transit node maintains a buffer associated with each flow. The structure of this buffer is shown in Figure 5. The buffer has a fixed size and wraps around once it fills up. The data input rate at each in-transit node is the amount of data queued at the buffer per second and the buffer drainage rate is proportional to the network connectivity of the outgoing link. The buffering algorithm at the node is reactive in that it attempts to dynamically adjust to the buffer input and buffer drainage rates. It does this by aggregating the blocks of data that have accumulated since the start of the transfer and transfers this aggregated block of data in the next transmission.

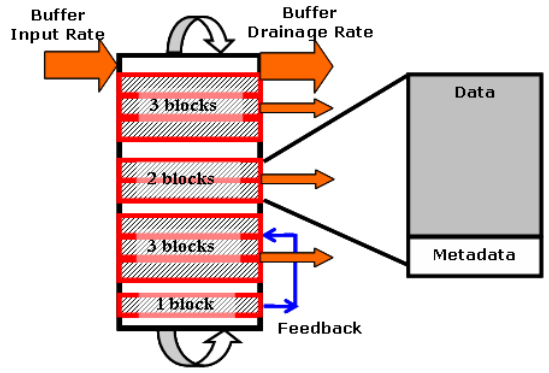

Fig. 5. Adaptive buffering at the in-transit node.

The size of the block transferred thus depends on the network connectivity and the transfer time of the previous transfer. Data transmission is multi-threaded and the number of transmission threads is controlled dynamically. Depending on the data input and drainage rates, the following situations can occur:

- Input rate exceeds drainage rate: In this situation the node attempts to maximize the data sent out by increasing the level of multi-threading at the transmission layer and improves throughput.
- Input rate is approximately equal to the drainage rate: In this situation new data accumulates in the buffer during each transfer. The first transfer will be the first data block queued, and the subsequent transfers will consist of blocks aggregated during the previous transfer. The buffer management scheme subsequently achieves equilibrium on the number of blocks transferred.
- Input rate is smaller than the drainage rate: In this situation, if the buffer manager encounters an empty buffer, it waits until more data is queued.

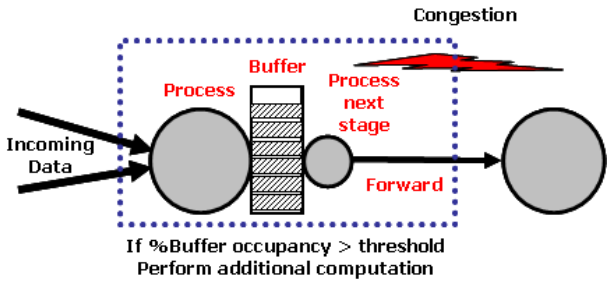

Fig. 6. Adaptive processing of data at in-transit nodes in response to network congestions.

The operation of an in-transit node is a follows. Each incoming data block is first processed, then queued in the buffer and finally forwarded to the next stage. Thus, the time spent by a data block at each in-transit node is thus the sum of the processing time (t_p), buffering time (t_{buff}) and forwarding time (t_f). During congestion, t_{buff} can sharply increase in relation to t_p and t_f. Since congestion can cause buffer overflows and loss of data at the in-transit nodes. In this case, rather the node attempts to further process the data block. The heuristic used is based on %Buffer Occupancy, i.e. the %data blocks stored in the buffer - when a node's buffer occupancy exceeds a certain threshold; the node decides to perform additional computation on the data blocks. This is illustrated in Figure 6. The rationale is that subsequent in-transit nodes downstream or the sink will then have to perform a smaller processing, which will offset the increased latency due to congestion.

C. Cooperative Self-Management: Coupling Application Level and In-Transit Management

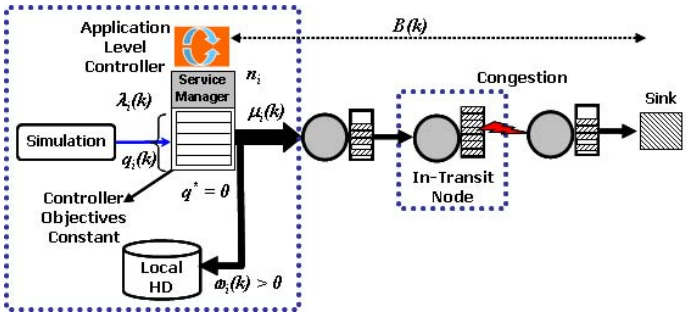

Fig. 7. Application level management in response to network congestions (without coupling).

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

In the standalone case as illustrated in Figure 7, if application level management was used in isolation, the application level controller would detect the congestion by observing a decrease of parameter $B(k)$, and it would advise the service manager to increase $\omega_i(k)$ and decrease $\mu_i(k)$, i.e., to reduce the amount of data sent on the network and increase the amount of data written to the local storage thereby avoiding data loss. While this would eventually reduce the congestion in the data path, it would require that the data blocks written to the local storage be manually transferred to and processed at the sink.

However in the coupled scenario (see Figure 8), the in-transit node signals the controller at the source in response to local congestion that it detects by observing its buffer occupancy and sends it information about its current buffer size. This allows the application level controller to detect congestion more rapidly, rather than have to wait until the congestion propagates back to the source, and in response, it increases its $q_i(k)$ (or in turn q^*) to a value higher than zero so as to throttle items in its buffer till the congestion at the in-transit nodes is relieved. This, in turn, reduces the amount of data that is written to the local disk at the source.

IV. Implementation and Experiments

This section presents experiments using the cooperative self-managing data streaming service as part of a fusion

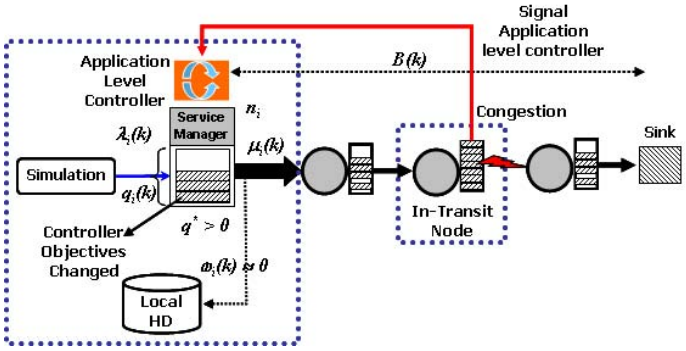

Fig. 8. Cooperative end-to-end management - in-transit node signals application controller about network congestions (with coupling).

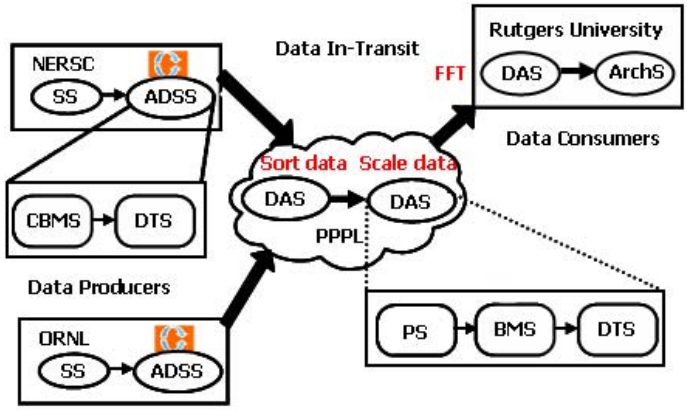

Fig. 9. The fusion simulation workflow used in the experiments.

workflow. The overall application setup is shown in Figure 9. It consists of the Simulation Service (SS), i.e., the GTC fusion simulation, which runs at NERSC (CA) and ORNL (TN), and streams data for analysis to PPPL (NJ) and final data archiving at Rutgers University (NJ). The simulation service (SS) executes on 32 to 256 processors on "Seaborg", an IBM SP machine at NERSC, and on 256 processors on "RAM", an SGI Altix machine. The Autonomic (self-managing) Data Streaming Service (ADSS) is co-located with the SS at NERSC and ORNL. The in-transit processing is performed by the Data Analysis Service (DAS) located at the in-transit nodes at PPPL and Rutgers. Data Archiving Service (ArchS) is also located at Rutgers which is referred to as the sink or data consumer. Three in-transit nodes were used in these experiments. These included 32 AMD Athlon MP 2100+ processors ("gridn" cluster), 4 dual-core AMD Opteron processors ("portalx" cluster) both located at PPPL and a 64 processor Intel Pentium (1.70GHz) Beowulf cluster ("Frea") located at Rutgers. Note that there is a 155 Mbps (peak) ESNET [4] connection between PPPL and NERSC and a 100 Mbps network connection between PPPL and Rutgers.

The ADSS service consists of a Controller based Buffer Management Service (CBMS), which contains an LLC online controller, and a Data Transfer Service (DTS). The controller interval for the CBMS was set to 80 seconds based on the data generation rates at the simulation end [1]. DTS uses a generic high performance transfer library for transferring data from simulation machines and is based on Logistical Networking (LN) [5].

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

A. Normal operation of DAS without congestion

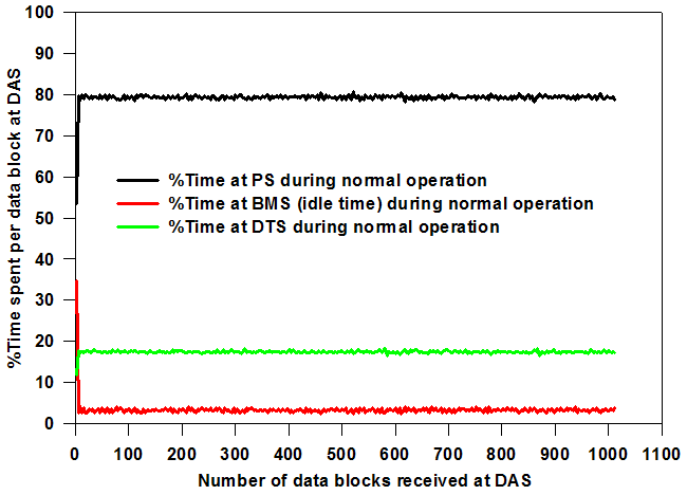

Fig. 10. Breakup of the %time spent at the each of the services comprising the DAS per data block.

This experiment evaluates the behavior of DAS at the in-transit nodes during normal operation, i.e., when there is no congestion. Figure 10 plots the average relative times spent per data block on each of the three component services, i.e., processing (PS), buffering (BMS) and forwarding (DTS). As seen from the figure, processing time (i.e., PS) is 80% on average, buffering time (i.e., BMS) is 3.2% on average, and forwarding time (i.e., DTS) is 17.8% on average. Buffering time mainly denotes the idle time for the data block in the DAS. Note that during the initial phases of the experiment, it is observed that the BMS time is significantly higher because of initial buffer warm up. This experiment provides the baseline for the experiments presented below.

B. Operation of the DAS during congestion but without adaptation

In this experiment, congestion was introduced between PPPL and Rutgers using the Trickle library [6], and the experiments conducted above were repeated. Figure 11 once again plots the average relative times spent per data block on

each of the three component services, i.e., processing (PS), buffering (BMS) and forwarding (DTS). The plots show that during congestion, the forwarding time increases significantly when compared to the normal operation case (i.e., Figure 10) and accounts for 41.64% of the total time. The buffering time (i.e., BMS) also increases as expected. Since there is no adaptation, the processing component (i.e., PS) remains the same but only accounts for 33% of the total time in this case.

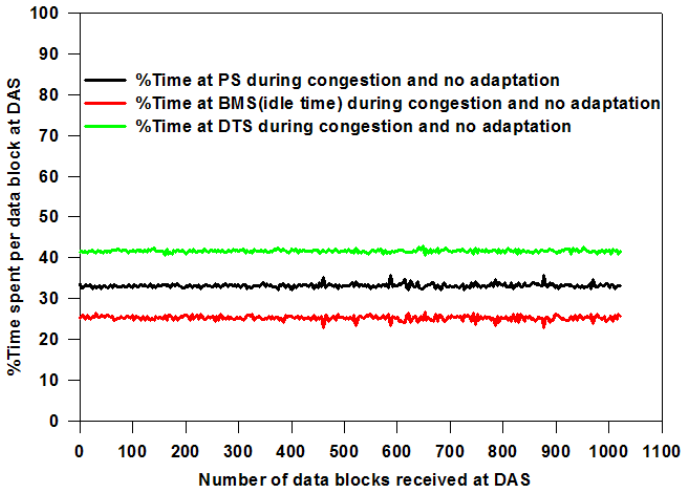

Fig. 11. Breakup of %time spent at the each of the services comprising the DAS per data block during congestion and no adaptation.

C. Operation of DAS during congestion with adaptation

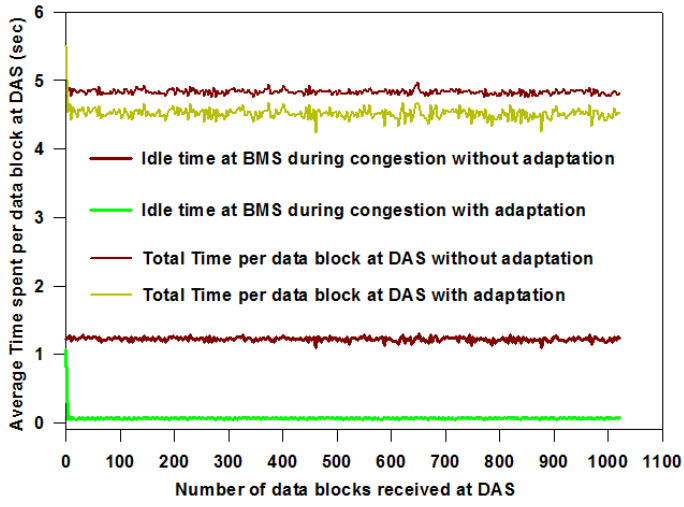

Fig. 12. Effects of adaptation on DAS during congestion - buffering or idle time reduced significantly.

This experiment modifies the experiment above to introduce adaptation at the in-transit nodes, i.e., the DAS service adaptively processes data in its buffers when it observes the %buffer occupancy is above 60%. As seen in Figure 13, the buffering (BMS) time decreases and the processing (PS) time increases correspondingly as expected. The adaptation does not effect the forwarding (DTS) time. The effects of the adaptation can be seen in Figure 12. In this figure, the buffering (BMS) time (thick lines in the graph) reduces from an average of 1.2 seconds in the case without adaptation to an average of 0.06 seconds with adaptation. The overall time per data block in the DAS is slightly reduced from 4.83 seconds to 4.53 seconds as data blocks would have to be written to high latency local storage without adaptation.

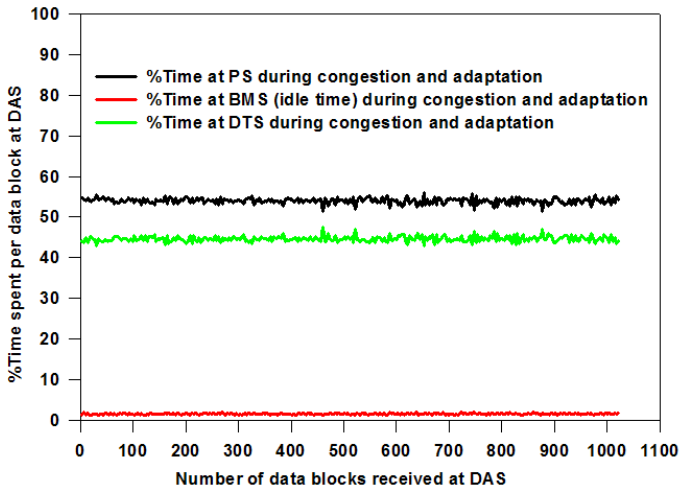

Fig. 13. Breakup of %time spent at the each of the services comprising the DAS per data block during congestion with adaptation.

D. Operation of ADSS with and without coupling

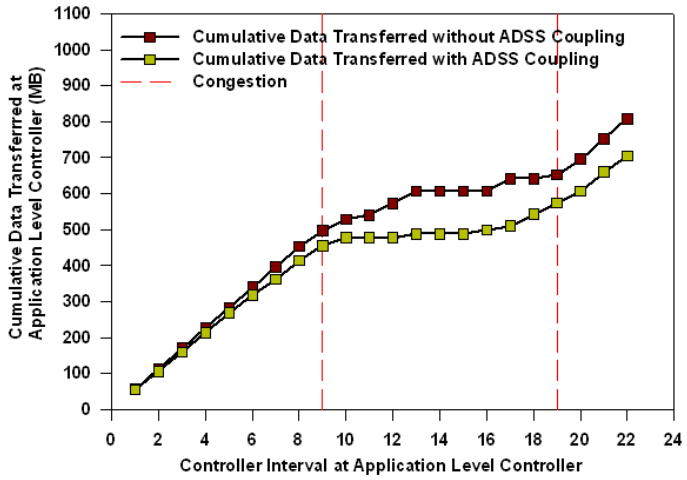

Fig. 14. ADSS behaviour with and without coupling.

This experiment evaluates the end-to-end behavior of the application level ADSS service with and without cooperative management and coupling with the in-transit DAS service. The cumulative data transferred for different controller intervals for the two cases are plotted in Figure 14. Since congestion events sent by the in-transit nodes cause ADSS to buffer data blocks rather than having them written to local storage, the

effective cumulative data transferred during congestion (i.e., controller intervals 9-20) drops. Figure 15 plots the average

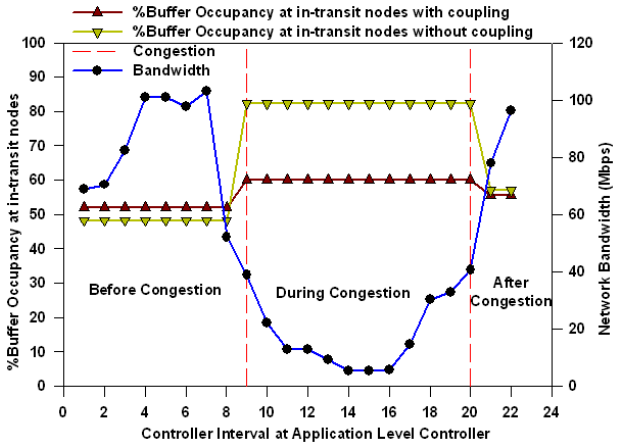

Fig. 15. Average %Buffer Occupancy at the in-transit nodes with coupling.

%buffer occupancy at an in-transit node (averaged over the three in-transit nodes used in the experiments) before, during and after congestion for this experiment. The average %buffer occupancy before the congestion is between 48.2% and 51%, which corresponds to normal operation (slight increase is due to the overheads of adaptation). During congestion, ADSS decides to throttle data blocks in response to congestion events from the in-transit nodes. This causes the average %buffer occupancy to decrease to about 60.8%. Without throttling and coupling, the average %buffer occupancy is significantly higher above 80%. Higher buffer occupancies at the in-transit nodes may lead to failures and result in data being dropped, and can impact the QoS at the sink. After the congestion clears and ADSS stops throttling data, the average %buffer occupancy at the in-transit nodes resets to around 50-57%.

E. Effect of adaptations at in-transit nodes on the quality of data received at sink

This experiment measures the quality of data received at the sink, in terms of the number of processing functions completed, with congestion and with and without in-transit adaptations. The higher the number of processing functions completed, the higher the quality and utility of the data to the sink. The quality of data without adaptations is plotted in Figure 16. It can be seen from the plot that during congestion, the cumulative amount of data received at the sink with 3 processing functions (PS) applied is 0 MB, while the cumulative amount of data received with 2 processing functions (PS) applied is around 300 MB. In contrast, when adaptation at the in-transit nodes are turned on in Figure 17, the cumulative amount of data received at the sink with 3 processing functions (PS) applied is around 232 MB. Higher data quality can save significant time at the sink. For example, if the average processing time per data block is 1.6 sec, adaptations save about 372 sec (approx. 6 minutes) of processing time at the sink.

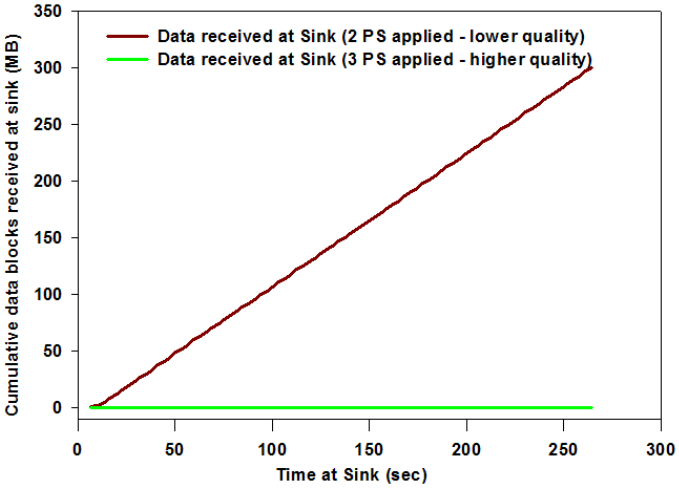

Fig. 16. Quality of data received at sink during congestion without adaptation at in-transit nodes.

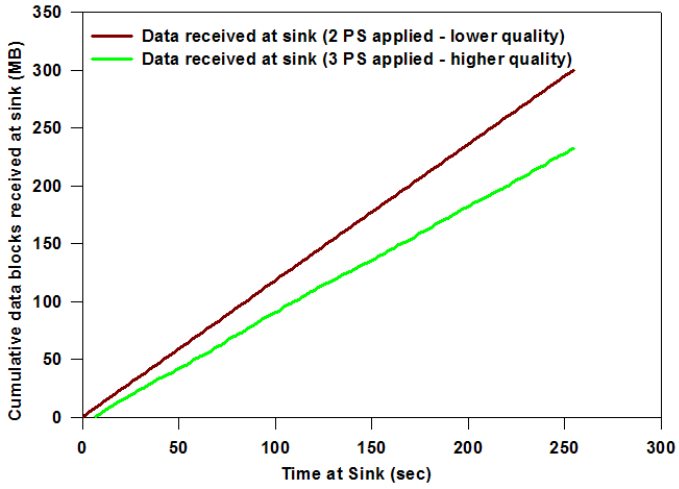

Fig. 17. Quality of data received at sink during congestion with adaptation at in-transit nodes.

F. Effectiveness of end-to-end cooperative management

This experiment measures the cumulative amount of data that is not delivered on time to the sink with only application level management and with end-to-end cooperative management. This is plotted in Figure 18. In all cases, when there is no congestion, all data blocks reach the sink. However, when there is congestion, if only application level management is used, about 399 MB does not reach the sink. When cooperative management is used, this drops to around 294 MB.

V. Related Work

Data Grids [7] and related research efforts such as SRM [8] have focused on the management and transport of large volumes of data for visualization and analysis. Traditionally data movement in Grid systems has been done using specialized protocols such as GridFTP [9], Internet Backplane Protocol (IBP) [10], bbcp [11] and SABUL [12], which define file management and transfer protocols for general-purpose secure,

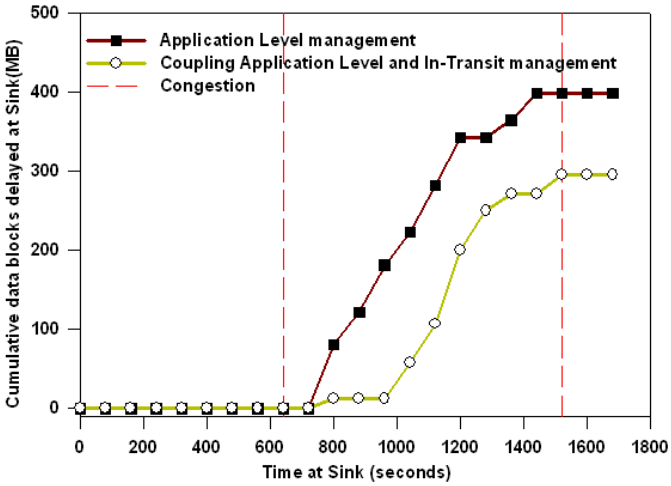

Fig. 18. Cumulative amount of data that does not reach the sink in time with and without cooperative management.

reliable data movement in Data Grids. These efforts focus only on data movement and not on in-transit data manipulations, and are complementary to this work. There is also some existing work addressing scientific data manipulation, such as DataCutter [13], which focuses more the partitioning and distribution of out-of-core computation, rather than scheduling, runtime and QoS management. Research efforts addressing Grid workflows, such as Pegasus [14], Sphinx [15], GridBus [16], GridAnt [17] and myGrid [18], focus on constructing end-to-end Grid applications. These efforts have addressed QoS issues related to workflow execution using cost based scheduling, brokering, and negotiation using Grid Quality of Service Management (G-QoSm). The work presented in this paper is complementary to these efforts and can be used in combination with them. The systems that are most related to this work are adaptive Grid workflow systems, such as Active Buffering [19] and Autoflow [20], which address issues of data streaming and/or in-transit processing. Our approach however differs from these in that it develops a cooperative two level approach that is specifically targeted to address both data streaming and in-transit manipulations for Grid workflows.

VI. Conclusion

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

Acknowledgements

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

References

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

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

[3] H. Liu, "Accord: A Programming System for Autonomic Self-Managing Applications," Ph.D. dissertation, Rutgers University, 2005.

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

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

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

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

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

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

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

[11] A. Hanushevsky, "bbcp peer to peer cp program," 2002. [Online]. Available: http://www.slac.stanford.edu/~abh/bbcp/

[12] Y. Gu and R. Grossman, "SABUL: A Transport Protocol for Grid Computing," *Journal of Grid Computing*, vol. 1, no. 4, pp. 377–386, 2003.

[13] M. Beynon, C. Chang, U. Catalyurek, T. Kurc, A. Sussman, R. Andrade, H.and Ferreira, and J. Saltz, "Processing Large-Scale Multi-Dimensional Data in Parallel and Distributed Environments," *Parallel data-intensive algorithms and applications*, vol. 28, no. 5, pp. 827–859, 2002.

[14] G. Singh, C. Kesselman, and E. Deelman, "Optimizing Grid-based Workflow Execution," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 201–219, September 2005.

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

[16] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 6, pp. 685 – 699, May 2006.

[17] G. v. Laszewski, K. Amin, and et. al, "GridAnt: A Client-Controllable Grid Workflow System," in *37th Hawaii International Conference on System Science (HICSS'04)*, Waikoloa, Big Island, Hawaii, 2004, p. 10.

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

[19] X. Ma, J. Lee, and M. Winslett, "High-Level Buffering for Hiding Periodic Output Cost in Scientific Simulations," *IEEE Transactions Parallel Distributed Systems*, vol. 17, no. 3, pp. 193–204, March 2006.

[20] Z. Cai, V. Kumar, and K. Schwan, "IQ-Paths: Predictably High Performance Data Streams Across Dynamic Network Overlays," *Journal of Grid Computing*, vol. 5, no. 2, pp. 129–150, June 2007.