Chapter 5

High Throughput Data Movement

Scott Klasky,[1] Hasan Abbasi,[2] Viraj Bhat,[3] Ciprian Docan,[3] Steve Hodson,[1] Chen Jin,[1] Jay Lofstead,[2] Manish Parashar,[3] Karsten Schwan,[2] and Matthew Wolf[2]

[1]*Oak Ridge National Laboratory*
[2]*Georgia Institute of Technology*
[3]*Rutgers, The State University of New Jersey*

Contents

5.1 Introduction

In this chapter, we look at technology changes affecting scientists who run data-intensive simulations, particularly concerning the ways in which these computations are run and how the data they produce is analyzed. As computer systems and technology evolve, and as usage policy of supercomputers often permit very long runs, simulations are starting to run for over 24 hours and produce unprecedented amounts of data. Previously, data produced by supercomputer applications was simply stored as files for subsequent analysis, sometimes days or weeks later. However, as the amount of the data becomes very large and/or the rates at which data is produced or consumed by supercomputers become very high, this approach no longer works, and high-throughput data movement techniques are needed.

Consequently, science-driven analytics over the next 20 years must support high-throughput data movement methods that shield scientists from machine-level details, such as the throughput achieved by a file system or the network bandwidth available to move data from the supercomputer site to remote machines on which the data is analyzed or visualized. Toward this end, we advocate a new computing environment in which scientists can ask, "What if I increase the pressure by a factor of 10?" and have the analytics software run the appropriate methods to examine the effects of such a change without any further work by the scientist. Since the simulations in which we are interested run for long periods of time, we can imagine scientists doing in-situ visualization during the lifetime of the run. The outcome of this approach is a paradigm shift in which potentially plentiful computational resources (e.g., multicore and accelerator technologies) are used to replace scarce I/O (Input/Output) capabilities by, for instance, introducing high-performance I/O with visualization, without introducing into the simulation code additional visualization routines.

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

In order to illustrate the high-throughput data requirements associated with data-intensive computing, we describe next in some detail an example of a real, large-scale fusion simulation. Fusion simulations are conducted in order to model and understand the behavior of particles and electromagnetic waves in tokomaks, which are devices designed to generate electricity from controlled nuclear fusion that involves the confining and heating of a gaseous plasma by means of an electric current and magnetic field. There are a few small devices already in operation, such as DIII-D [1] and NSTX [2], and a large device in progress, ITER [3], being built in southern France.

The example described next, and the driver for the research described in this chapter, is the gyrokinetic toroidal code (GTC) [4] fusion simulation that scientists ran on the 250+ Tflop computer at Oak Ridge National Laboratory (ORNL) during the first quarter of 2008. GTC is a state-of-the-art global fusion code that has been optimized to achieve high efficiency on a single computing node and nearly perfect scalability on massively parallel computers. It uses the particle-in-cell (PIC) technique to model the behavior of particles and electromagnetic waves in a toroidal plasma in which ions and electrons are confined by intense magnetic fields. One of the goals of GTC simulations is to resolve the critical question of whether or not scaling in large tokamaks will impact ignition for ITER.

In order to understand these effects and validate the simulations against experiments, the scientists will need to record enormous amounts of data. The particle data in the PIC simulations is five-dimensional, containing three spatial dimensions and two velocity dimensions. The best estimates are that the essential information can be 55 GB of data written out every 60 seconds. However, since each simulation takes 1.5 days, and produces roughly 150 TB of data (including extra information not included in our previous calculation), it is obvious that there will not be enough disk space for the next simulation scheduled on the supercomputer unless the data is archived on the high-performance storage system, HPSS [32], while the simulation is running. Moving the data to HPSS, running at 300 MB/sec still requires staging simulations, one per week. This means that runs will first need to move the data from the supercomputer over to a large disk. From this disk, the data can then move over to HPSS, at the rate of 300 MB/sec.

Finally, since human and system errors can occur, it is critical that scientists monitor the simulation during its execution. While running on a system with 100,000 processors, every wasted hour results in 100,000 wasted CPU hours. Obviously we need to closely monitor simulations in order to conserve the precious resources on the supercomputer, and the time of the application scientist after a long simulation. The general analysis that one would do during a simulation can include taking multidimensional FFTs (fast fourier transforms) and looking at correlation functions over a specified time range, as well as simple statistics. Adding these routines directly to the simulation not only complicates the code, but it is also difficult to make all of the extra routines scale as part of the simulation. To summarize, effectively running the large simulations to enable cutting-edge science, such as the GTC fusion simulations described above, requires that the large volumes of data generated must be (a) moved from the compute nodes to disk, (b) moved from disk to tape, (c) analyzed during the movement, and finally (d) visualized, all while the simulation is running. Workflow management tools can be used very effectively for this purpose, as described in some detail in Chapter 13.

In the future, codes like GTC, which models the behavior of the plasma in the center of the device, will be coupled with other codes, such as XGC1 [5], which models the edge of the plasma. The early version of this code, called XGC0, is already producing very informative results that fusion experimentalists are beginning to use to validate against experiments such as DIII-D and NSTX. This requires loose coupling of the kinetic code, XGC0, with GTC and other simulation codes. It is critical that we monitor the XGC0 simulation results and generate simple images that can be selected and displayed while the simulation is running. Further, this coupling is tight, that is, with strict space and time constraints, and the data movement technologies must be able to support such a coupling of these codes while minimizing programming effort. Automating the end-to-end process of configuring, executing, and monitoring of such coupled-code simulations, using high-level programming interfaces and high-throughput data movement is necessary to enable scientists to concentrate on their science and not worry about all of the technologies underneath.

Please provide full term.

Clearly a paradigm shift must occur for researchers to dynamically and effectively find the needle in the haystack of data and perform complex code coupling. Enabling technologies must make it simple to monitor and couple codes and to move data from one location to another. They must empower scientists to ask "what if" questions and have the software and hardware infrastructure capable of answering these questions in a timely fashion. Furthermore, effective data management is not just becoming important—it is becoming absolutely essential as we move beyond current systems into the age of exascale computing. We can already see the impact of such a shift in other domains; for example, the Google desktop has revolutionized desktop computing by allowing users to find information that might have otherwise gone undetected. These types of technologies are now moving into leadership-class computing and must be made to work on the largest analysis machines. High-throughput end-to-end data movement is an essential part of the solution as we move toward exascale computing. In the remainder of the chapter, we present several efforts toward providing high-throughput data movement to support these goals.

The rest of this chapter will focus on the techniques that the authors have developed over the last few years for high-performance, high-throughput data movement and processing. We begin the next section with a discussion of the Adaptable IO System (ADIOS), and show how this can be extremely valuable to application scientists and lends itself to both synchronous and asynheronous data movement. Next, we describe the Georgia Tech DataTap method underlying ADIOS, which supports high-performance data movement. This is followed with a description of the Rutgers DART (decoupled and asynheronous remote transfers) method, which is another method that uses remote direct memory access (RDMA) for high-throughput asynchronous data transport and has been effectively used by applications codes including XGC1 and GTC. Finally, we describe mechanisms, such as autonomic management techniques and in-transit data manipulation methods, to support complex operations over the LAN and WAN.

5.2 High-Performance Data Capture

A key prerequisite to high-throughput data movement is the ability to capture data from high-performance codes with low overheads such that data movement actions do not unnecessarily perturb or slow down the application execution. More succinctly, data capture must be flexible in the overheads and perturbation acceptable to end-user applications. This section first describes the ADIOS API and design philosophy and then describes two specific examples of data capture mechanisms, the performance attained by them, and the overheads implied by their use.

Please provide full term.

5.2.1 Asynchronous Capture of Typed Data

Even with as few as about 10,000 cores, substantial performance degradation has been seen due to inappropriately performed I/O. Key issues include I/O systems difficulties in dealing with large numbers of writers into the same file system, poor usage of I/O formats causing metadata-based contention effects in I/O subsystems, and synchronous I/O actions unable to exploit communication/computation overlap. For example, when a simulation attempts to open, and then write one file per processor, the first step is to contact the metadata service of the parallel file system, issuing tens of thousands of requests at once. This greatly impacts the speed of I/O. Furthermore, scientific data is generally written out in large bursts. Using synchronous I/O techniques makes the raw speed to write this data the limiting factor. Therefore, if a simulation demands that the I/O rate take less than 5 percent of the calculation cost, then the file system must be able to write out, for example, 10 TB of data every 3600 seconds (generated in burst mode at a rate of 56 GB/sec). Using asynchronous techniques instead would only require a sustained 2.8 GB/sec write in the same case.

A first step to addressing these problems is to devise I/O interfaces for high-performance codes that can exploit modern I/O techniques while providing levels of support to end users that do not require them to have intimate knowledge of underlying machine architectures, I/O, and communication system configurations.

The Adaptable I/O System, ADIOS, is a componentization of the I/O layer. It provides the application scientist with easy-to-use APIs, which are almost as simple as standard FORTRAN write statements. ADIOS separates the metadata "pollution" away from the API, and allows the application scientist to specify the variables in their output in terms of groups. For example, let's suppose that a user has a variable, zion, which is associated with the ion particles of the plasma. The variable has the units of m/s (meters/second), and has the long name of ion parameters. Conventionally, all of this metadata must be written in the code, which involves placing these statements inside the Fortran/C code. In the ADIOS framework, the application scientist creates an XML file that contains this information, along with the specification of the method for each group, such as MPI-IO, or POSIX. The method declarations can be switched at runtime and allow the scientist to change from POSIX I/O, to MPI-IO, to asynchronous methods such as the DataTap services [6] and the DART system [7] described below. By allowing the scientist to separate out the I/O implementation from the API, users are allowed to keep their code the same and only change the underlying I/O method when they run on different computers. Another advantage of specifying the information in this manner is that the scientist can just maintain one write statement for all of the variables in a group, thus simplifying their programs. This system also allows the user to move away from individual write statements, and as a result, the system can buffer the data and consequently write large blocks of data, which works best in parallel file systems. A small example of an XML file is as follows.

Please provide full term.

```
< ioconfig >

  <datatype name=''restart''>

      <scalar  name=''mi''  path =''/ param''  type='' integer ''/>

      <dataset  name=''zion''  type='' real ''  dimensions=''n ,1:4,2, mi''/>
```

<data− attribute name=‘‘units ’’ path =‘‘/ param’’ value=‘‘m/s’’/>

<data− attribute name=‘‘long{_}name’’ path=‘‘/param’’ value=‘‘ion parameters ’’/>

< /**datatype** >

<**method** priority=‘‘1’’ method=‘‘DATATAP’’ iterations=‘‘1’’ type=‘‘diagnosis’’> srv=ewok001.ccs.ornl . gov</**method**>

< /**ioconfig** >

Most importantly, however, ADIOS can provide such methods with rich metadata about the data being moved, thereby enabling the new paradigms for high-throughput data movement. These new paradigms include: (1) compact binary data transmission using structure information about the data (e.g., for efficient interpretation of data layout and access to and manipulation of select data fields): (2) the ability to operate on data as it is being moved (e.g., for online data filtering or data routing); and (3) the ability to use appropriate underlying *transport mechanisms* (e.g., such as switching from MPI-I/O to POSIX, to netCDF, to HDF-5). Furthermore, we envision building a code-coupling framework extending the ADIOS APIs that will allow scientists to try different mathematical algorithms by simply changing the metadata. Beyond providing information about the structure of the data, ADIOS also has built-in support for collecting and forwarding to the I/O subsystem key performance information, enabling dynamic feedback for scheduling storage-related I/O, the external configuration of data collection and storage/processing mechanisms, and value-added, additional in-flight and offline/near-line processing of I/O data. For example, specifying that the data be reduced in size (using a program that is available where the data is) before the data is written to the disk.

ADIOS encodes data in a compact, tagged, binary format for transport. This can either be written directly to storage or parsed for repackaging in another format, such as HDF-5 or netCDF. The format consists of a series of size-marked elements, each with a set of tags-values pairs to describe the element and its data. For example, an array is represented by a tag for a name, a tag for a data path for HDF-5 or similar purposes, and a value tag. The value tag contains rank of the array, the dimensional magnitude of each rank, the data type, and the block of bytes that represent the data. In the previous example where we showed the XML data markup, the large array written during the restarts for GTC is zion. Zion has rank=4, the dimensions are (n $\times$ 4 $\times$ 2 $\times$ mi) on each process, and the block of bytes will be 4*n*4*2*mi bytes. The remainder of this section demonstrates the utility of ADIOS for driving future work in high-throughput data movement, by using it with two different asynchronous data capture and transport mechanisms: the Rutgers DART system and the Georgia Tech LIVE DataTap system. Before doing so, we first summarize some of the basic elements of ADIOS.

ADIOS exploits modern Web technologies by using an external XML configuration file to describe all of the data collections used in the code. The file describes for each element of the collection the data types, element paths (similar to HDF-5 paths), and dynamic and static array sizes. If the data represents a mesh, information about the mesh, as well as the global bounds of an array, and the ghost regions[1] used in the MPI programming is encoded in the XML structure. For each data collection, it describes the transport mechanism selection and parameters as well as pacing information for timing the data transmissions. With this information, the ADIOS I/O implementation can then control when, how, and how much data is written at a time, thereby affording efficient overlapping with computation phases of scientific codes and proper pacing to optimize the write performance of the storage system.

A basic attributes of ADIOS is that it de-links the direct connection between the scientific code and the manipulation of storage, which makes it possible to add components that manipulate the I/O data outside the realm of the supercomputer's compute nodes. For example, we can inject filters that generate calls to visualization APIs like Visit [8], route the data to potentially multiple destinations in multiple formats, and apply data-aware compression techniques.

Several key high-performance computing (HPC) applications' driving capacity computing for petascale machines have been converted to using ADIOS, with early developments of ADIOS based on two key HPC applications: GTC (a fusion modeling code) and Chimera (an astrophysics supernova code) [9]. Prior to its use of ADIOS, GTC employed a mixture of MPI-IO, HDF-5, netCDF, and straight Fortran I/O; and Chimera used straight Fortran I/O routines for writing binary files. Both of these codes provided different I/O requirements that drove the development of the API. Specifically, in GTC, there are seven different data formats, corresponding to various restart, diagnostic, and analysis values. Some of the data storage format requirements, such as combining some data types together in the same file, represent a good exercise of the capabilities of the ADIOS API. Chimera exercises the ADIOS API in three different ways. First, it contains approximately 475 different scalars and arrays for a single restart format. Second, this data ideally needs to be stored in both an efficient binary format and a readable text format. Third, the large number of elements encouraged the development of an experimental data reading extension for the ADIOS API that follows similar API semantics as used for writing, and leverages the writing infrastructure as much as possible. A simple code fragment showing how ADIOS is used is presented below:

```
call adios{\_}init ('config.xml')

...

! do main loop

call adios{\_}begin{\_}calculation ()
```

[1]Ghost regions are regions that overlap adjacent grid cells.

```
! do non-communication work

call adios{\_}end{\_}calcuation ()

...

! perform restart write

...

! do communication work

call adios{\_}end{\_}iteration ()

! end loop

...

call adios{\_}finalize ()
```

Adios_init () initiates parsing of the configuration file generating all of the internal data type information, configures the mechanisms described above, and potentially sets up the buffer. Buffer creation can be delayed until a subsequent call to adios_allocate_buffer if it should be based on a percentage of free memory or other allocation-time-sensitive considerations.

Adios_begin_calculuation () and adios_end_calculation () provide the "ticker" mechanism for asynchronous I/O, providing the asynchronous I/O mechanism with information about the compute phases, so that the I/O can be performed at times when the application is not engaged in communications. The subsequent "end" call indicates that the code wishes to perform communication, ratcheting back any I/O use of bandwidth.

Adios_end_iteration () is a pacing function designed to give feedback to asynchronous I/O to gauge what progress must be made with data transmission in order to keep up with the code. For example, if a checkpoint/restart[2] is written every 100 iterations, the XML file may indicate an iteration count that is less than 100, to evacuate the data in order to accommodate possible storage congestion or other issues, such as a high demand on the shared network.

Adios_finalize () indicates the code is about to shut down and any asynchronous operations need to complete. It will block until all of the data has been drained from the compute node.

[2]When running simulations with many time steps, it is customary to write out checkpoint/restart data following a number of time steps, in case the computation needs to backtrack, thus avoiding repeating the computation from the start.

We describe next two asynchronous I/O mechanisms underlying ADIOS. Note that these mechanisms target current day supercomputers, which are typically composed of login nodes, I/O nodes, and compute nodes.

5.2.2 DataTaps and DataTap Servers

DataTap addresses the following performance issues for high-throughput data movement:

- Scaling to large data volumes and large numbers of I/O clients given limited I/O resources
- Avoiding excessive CPU and memory overheads on the compute nodes
- Balancing bandwidth utilization across the system
- Offering additional I/O functionality to the end users, including on-demand data annotation and filtering

In order to attain these goals, it is necessary to move structured rather than unstructured data, meaning, as expressed above in describing the ADIOS API, efficiency in data movement is inexorably tied to knowledge about the type and structure of the data being moved. This is because such knowledge makes it possible to manipulate data during movement, including routing it to appropriate sites, reorganizing it for storage or display, filtering it, or otherwise transforming it to suit current end-user needs. Next we describe the efficient, asynchronous data capture and transport mechanisms that underlie such functionality:

- **DataTaps** — flexible mechanisms for extracting data from or injecting data into HPC computations; efficiency is gained by making it easy to vary I/O overheads and costs in terms of buffer usage and CPU cycles spent on I/O and by controlling I/O volumes and frequency. DataTaps move data from compute nodes to DataTap servers residing on I/O nodes.
- **Structured data** — structure information about the data being captured, transported, manipulated, and stored enables annotation or modification both synchronously and asynchronously with data movement.
- **I/O graphs** — explicitly represent an application's I/O tasks as configurable overlay[3] topologies of the nodes and links used for moving and operating on data, and enable systemwide I/O resource management. I/O graphs start with the lightweight DataTaps on computational nodes; traverse arbitrary additional task nodes on the petascale machine (including compute and I/O nodes as desired); and "end" on storage, analysis, or data visualization engines. Developers use I/O graphs to flexibly and dynamically partition I/O tasks and concurrently

[3]Overlay networks are virtual networks of nodes on top of another physical network. For the I/O graphs, data moves between nodes in the I/O graph overlay via logical (virtual) links, whereas in reality it may traverse one or more physical links between the nodes in the underlying physical network.

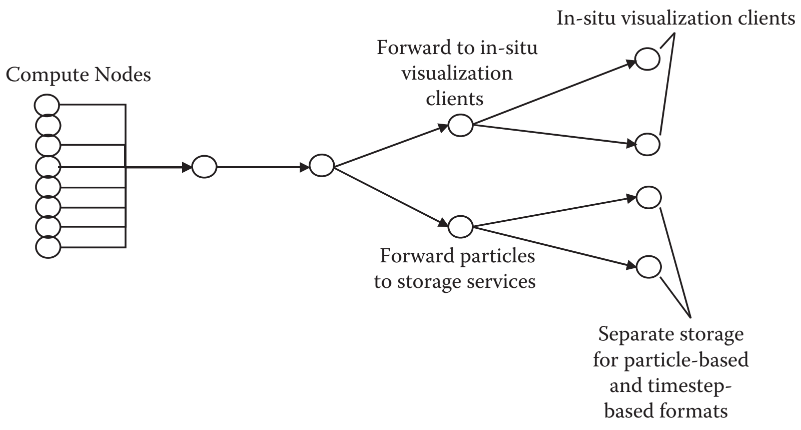

In-situ visualization clients
Forward to in-situ visualization clients
Compute Nodes
Forward particles to storage services
Separate storage for particle-based and timestep-based formats

Figure 5.1 I/O graph Example.

execute them across petascale machines and the ancillary engines supporting their use.

The simple I/O graphs shown in Figure 5.1 span compute to I/O nodes. This I/O graph first filters particles to only include interesting data — say, within some bounding boxes or for some plasma species. The filtering I/O node then forwards the particle data to other I/O nodes, which in turn forward particle information to in situ visualization clients (which may be remotely accessed), and to storage services that store the particle information two different ways — one in which the particles are stored based on the bounding box they fall in, and one in which the particles are stored based on the timestep and compute node in which the information was generated.

- **Scheduling techniques** dynamically manage DataTap and I/O graph execution, taking into account the I/O costs imposed on petascale applications.
- **Experimental results** attained with DataTaps and I/O graphs demonstrate several important attributes of I/O systems that benefit petascale machines. First, asynchronous I/O makes it possible to carry out I/O actions while massively parallel processor (MPP) computations are ongoing. This computation–I/O overlap improves throughput substantially, compared with the synchronous methods used by current file systems. Second, when performing I/O asynchronously, we demonstrated that it can scale without perturbing the applications running on compute nodes. For instance, sustained high-bandwidth data extraction (over 900 MB/s) has been achieved on the Cray XT4 without undue application perturbation and with moderate buffering requirements [6].

DataTaps Implementation A DataTap is a request-read service designed to address the difference between the available memory on typical MPP compute partitions and that on I/O and service nodes. We assume the existence of a large number of compute nodes producing data — DataTap clients — and a smaller number of I/O nodes receiving the data — DataTap servers. The DataTap client issues a data-available request to the DataTap server, encodes the data for transmission, and registers this buffer with the transport for remote read. For very large data sizes, the cost of encoding data can be significant, but it will be dwarfed by the actual cost of the data transfer [10–12]. On receipt of the request, the DataTap server issues a read call. The DataTap server feeds an I/O graph, which can replicate the functionality of writing the output to a file, or it can be used to perform "in-flight" data transformations.

The design and implementation of DataTap servers deal with several performance issues and constraints present on modern MPP machines. First, due to the limited amount of memory available on the DataTap server, the server only issues a read call if there is memory available to complete it. Second, since buffer space used by asynchronous I/O on compute nodes is limited, the server issues multiple read calls each time it operates. Third, the next generation of DataTap servers will install controls on the speed and timing of reading data from DataTap buffers. The goal is to prevent perturbation caused when I/O actions are performed simultaneously with internal communications of application code (e.g., MPI collectives). Additional constraints result from in-transit actions performed by I/O graphs; these are evaluated in our ongoing and future work.

The current implementation of DataTap leverages existing protocols (i.e., Cray Portals and InfiniBand RDMA). Since the abstraction presented to the programmer is inherently asynchronous and data driven, data movement can take advantage of data object optimizations like message aggregation, data filtering, or other types of in-transit data manipulations, such as data validation. In contrast, the successful paradigm of MPI-IO, particularly when coupled with a parallel file system, heavily leverages the file nature of the data target and utilizes the transport infrastructure as efficiently as possible within that model. That inherently means the underlying file system concepts of consistency, global naming, and access patterns will be enforced at higher levels as well. By adopting a model that allows for the embedding of computations within the transport overlay, it is possible to delay execution of or entirely eliminate those elements of the file object that the application does not immediately require. If a particular algorithm does not require consistency (as is true of some highly fault-tolerant algorithms), then it is not necessary to enforce it from the application perspective. Similarly, if there is an application-specific concept of consistency (such as validating a checkpoint file before allowing it to overwrite the previous checkpoint file), then that could be enforced, as well as all of the more application-driven specifications mentioned earlier.

DataTaps leverage extensive prior work with high-performance data movement, including (1) efficient representations of meta-information about data structure and layout (PBIO [13]); which enables (2) high performance and "structure-aware" manipulations on data in flight, carried out by dynamically deployed binary codes and using higher level tools with which such manipulations can be specified, termed XChange. [14]; (3) a dynamic overlay (i.e., the I/O graph) optimized for efficient data movement, where data fast path actions are strongly separated from the control actions

Please provide full term.

necessary to build, configure, and maintain the overlay[4] [15]; and (4) a lightweight object storage facility (LWFS [16]) that provides flexible, high-performance data storage while preserving access controls on data.

Because the DataTap API is not common to GTC or other current MPP applications, we use the ADIOS system to make DataTap (and structured stream) integration easier. By employing this API, a simple change in an entry in the XML file causes GTC, for example, to use synchronous MPI-IO, POSIX, our asynchronous DataTap servers, parallel-netCDF, HDF-5, NULL (no I/O performed), or other transports. Further, each data grouping, such as a restart versis diagnostic output, can use different transports, at no loss in performance compared with the direct use of methods like MPI-IO. The outcome is that integration details for downstream processing are removed from MPP codes, thereby permitting the user to enable or disable integration without the need for recompilation or relinking. A key property of structured streams preserved by ADIOS is the description of the structure of data to be moved in addition to extents or sizes. This makes it possible to describe semantically meaningful actions on data in ADIOS, such as chunking it for more efficient transport, filtering it to remove uninteresting data for analysis or display [17], and similar actions.

We describe next the use of DataTap with the GTC code, as an example. Once GTC had been modified to use ADIOS, it was configured to use the DataTap as an output transport. The DataTap uses some of the compute node memory, storage that would otherwise be available to GTC. This method allows the application to proceed with computation as the data is moved to the DataTap server. Once at the DataTap server, the data is forwarded into the I/O graph.

The GTC simulations use several postprocessing tasks. These include the visualization of the simulated plasma toroid with respect to certain parameters. We describe next how the visualization data is constructed using DataTap and the I/O graph. The visualization that has proven useful is a display of the electrostatic potential at collections of points in a cross-section of the simulated toroid, called poloidal planes. The poloidal plane is described by a grid of points, each of which has a scalar value — the electrostatic potential at that grid vertex. To construct an image, this grid can be plotted in two or three dimensions, with appropriate color values assigned to represent the range of potential values. The visualization can be constructed after the simulation is run by coordinating information across several output files. Using the I/O graph components, we can recover this information with minimal impact on the application and, equally importantly, while the application is running. This permits end users to rapidly inspect simulation results while the MPP code is executing.

The DataTap is comprised of two separate components, the server and the client. The DataTap server operates on the I/O or service nodes, while the DataTap client is an I/O method provided to GTC through the ADIOS API. Because the number of compute nodes is so much greater than the number of service nodes, there is a

[4]Such control actions are referred to as the control layer for the overlay network.

corresponding mismatch between the number of DataTap clients and servers. To take advantage of asynchronicity, the DataTap client only issues a transfer request to the DataTap server instead of sending the entire data packet to the server. Also, to enable asynchronous communication the data is buffered before the data transfer request is issued. We use PBIO [18] to marshal the data into a buffer reserved for DataTap usage. The use of the buffer consumes some of the memory available to GTC but allows the application to proceed without waiting for I/O. The application only blocks for I/O while waiting for a previous I/O request to complete.

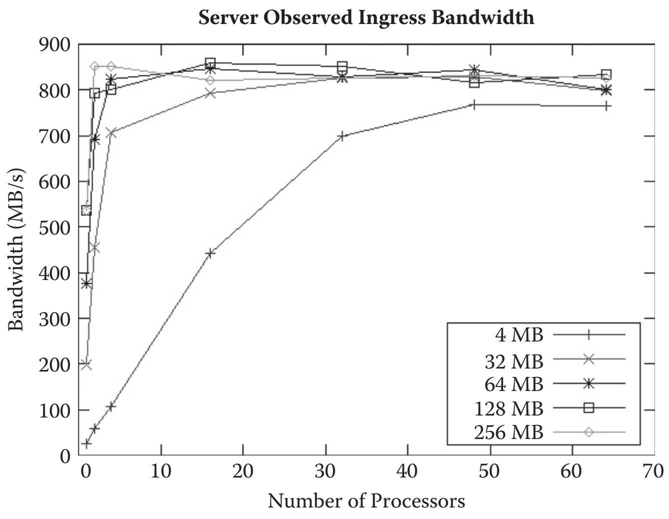
Once the DataTap server receives the request, it is queued up locally for future processing. The queuing of the request is necessary due to the large imbalance in the total size of the data to be transferred and the amount of memory available on the service node. For each request the DataTap server issues an RDMA read request to the originating compute node.

To maximize the bandwidth usage for the application, the DataTap server issues multiple RDMA read requests concurrently. The number of requests is predicated on the available memory at the service nodes and the size of the data being transferred. Also to minimize the perturbation caused by asynchronous I/O, the DataTap server uses a scheduling mechanism so as not to issue read requests when the application is actively using the network fabric. Once the data buffer is transferred over, the DataTap server sends the buffer to the I/O graph for further processing.

DataTap Evaluation To evaluate the efficiency and performance of the DataTap we look at the bandwidth observed at the DataTap server (at the I/O node). In Figure 5.2 we evaluate the scalability of our two DataTap implementations by looking at the maximum bandwidth achieved during data transfers. The InfiniBand DataTap (on a Linux Cluster) suffers a performance degradation due to the lack of a reliable datagram transport in our current hardware. However, this performance penalty only effects the first iteration of the data transfer, where connection initiation is performed. Subsequent transfers use cached connection information for improved performance. For smaller data sizes the Cray XT3 is significantly faster than the InfiniBand DataTap. The InfiniBand DataTap offers higher maximum bandwidth due to more optimized memory handling on the InfiniBand DataTap; we are currently addressing this for the Cray XT3 version.

In GTC's default I/O pattern, the dominant cost is from each processor's writing out the local array of particles into a separate file. This corresponds to writing out something close to 10% of the memory footprint of the code, with the write frequency chosen so as to keep the average overhead of I/O within a reasonable percentage of total execution time. As part of the standard process of accumulating and interpreting this data, these individual files are then aggregated and parsed into time series, spatially bounded regions, and so forth, depending on downstream needs.

To demonstrate the utility of structured streams in an application environment, we evaluated GTC on a Cray XT3 development cluster at ORNL with two different input set sizes. For each, we compared GTCs runtime for three different I/O configurations: no data output, data output to a per-mpi-process Lustre file, and data output using a DataTap (Table 5.1). We observed a significant reduction in the overhead caused by



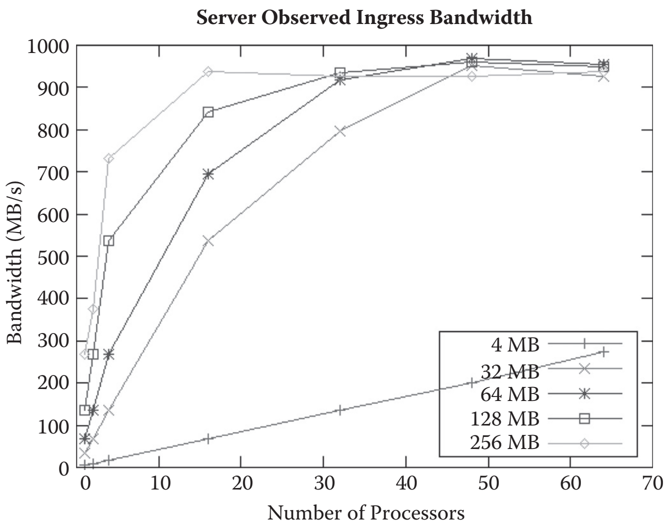


Figure 5.2 DataTap performance, on the left from the Cray XT3, and on the right from Infiniband.

the data output as the input set size increases, from about 9% on Lustre to about 3% using DataTap.[5]

The structured stream is configured with a simple I/O graph: DataTaps are placed in each of the GTC processes, feeding out asynchronously to an I/O node. From the I/O node, each message is forwarded to a graph node where the data is partitioned

[5]We define the I/O overhead as (time with I/O − total time with no I/O)/total time with no I/O.

TABLE 5.1 Comparison of GTC run times on the ORNL Cray XT3 development machine for two input sizes using different data output mechanisms

Run Parameters	Time for 100 iterations (582,410 ions)	Time for 100 iterations (1,164,820 ions)
No Output	213	422
Lustre	232	461
DataTap	220	435

into different bounding boxes. Once the data is received by the DataTap server, we filter the data based on the bounding box and then transfer the data for visualization. Copies of both the whole data and the multiple small partitioned datasets are then forwarded on to the storage nodes. Since GTC has the potential of generating PBs of data, we find it necessary to filter/reduce the total amount of data. The time taken to perform the bounding box computation is 2.29s and the time to transfer the filtered data is 0.037s. In the second implementation we transfer the data first and run the bounding box filter after the data transfer. The time taken for the bounding box filter is the same (2.29s) but the time taken to transfer the data increases to 0.297s. The key is not the particular values for the two cases but rather the relationship between them, which shows the relative advantages and disadvantages. In the first implementation the total time taken to transfer the data and run the bounding box filter is lower, but the computation is performed on the DataTap server. This increases the server's request service latency. For the second implementation, the computation is performed on a remote node and the impact on the DataTap is reduced. The value of this approach is that it allows an end user to compose a utility function that takes into account the cost in time **at a particular location**. Since most centers charge only for time on the big machines, often times the maximum utility will show that filtering should be done on the remote nodes. If the transmission time to the remote site was to increase and slow down the computation more than the filtering time, higher utility would come from filtering the data before moving it. Thus, it is important that the I/O system be flexible enough to allow the user to switch between these two cases.

5.2.3 High-Speed Asynchronous Data Extraction Using DART

As motivated previously, scientific applications require a scalable and robust substrate for managing the large amounts of data generated and for asynchronously extracting and transporting them between interacting components. DART (decoupled and asynchronous remote transfers) [7] is an alternate design strategy to DataTap described above, and it is an efficient data transfer substrate that effectively addresses the requirements described above. Unlike DataTap, which attempts to develop an overall data management framework, DART is a thin software layer built on RDMA technology to enable fast, low-overhead, and asynchronous access to data from a running simulation, and support high-throughput, low-latency data transfers. The design and prototype implementation of DART using the Portals RDMA library on the

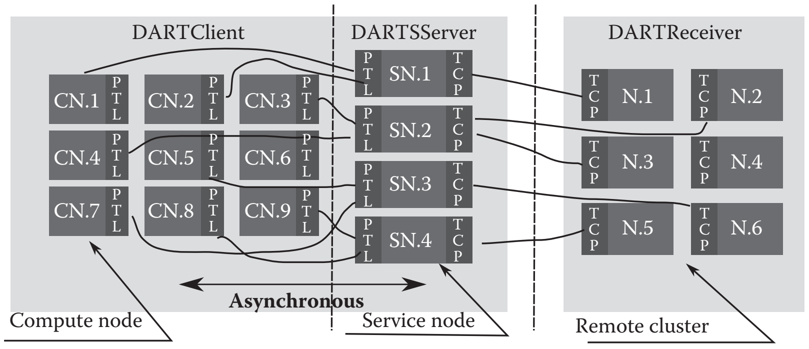


Figure 5.3 Architectural overview of DART.

Cray XT3/XT4 at ORNL are described next. DART has been integrated with the applications simulating fusion plasma in a Tokamak, described above, and is another key component of ADIOS.

The primary goal of DART is to efficiently manage and transfer large amounts of data from applications running on the compute nodes of an HPC system to the service nodes and remote locations, to support remote application monitoring, data analysis, coupling, and archiving. To achieve these goals, DART is designed so that the service nodes asynchronously extract data from the memory of the compute nodes, and so we offload expensive data I/O and streaming operations from the compute nodes to these service nodes. DART architecture contains three key components as shown in Figure 5.3: (1) a thin client layer (DARTClient), which runs on the compute nodes of an HPC system and is integrated with the application; (2) a streaming server (DARTSServer), which runs independently on the service nodes and is responsible for data extraction and transport; and (3) a receiver (DARTReceiver), which runs on remote nodes and receives and processes data streamed by DARTSServer.

A performance evaluation using the GTC simulation demonstrated that DART can effectively use RDMA technologies to offload expensive I/O operations to service nodes with very small overheads on the simulation itself, allowing a more efficient utilization of the compute elements, and enabling efficient online data monitoring and analysis on remote clusters.

5.2.4 In-transit services

In addition to the data movement and low-level data capture interfaces described above, applications that require high-throughput adaptive I/O must also depend on robust transport and specialization services. Such specialization services are required to perform "in-transit" data inspection and manipulation, including filtering, aggregation, or other types of processing actions that tune the data output to the current user- or application-specific requirements. The use of specialization services jointly with the basic data movement results in attainment of adaptive I/O services, needed to address the dynamism in application inputs and outputs, computational and communication loads and operating conditions, and end-user interests. We focus here on techniques for the autonomic tuning of these transports to provide the user-level quality of information and specification of utility that next-generation application data flows require. An example of this is a scientist who is particularly interested in one type of interatomic bond during a molecular dynamics simulation and who is, under bandwidth constraints, willing to rely on a specialized transport service that filters out simulation output not related to atoms involved in such bonds, or that gives those data outputs higher priority compared with other outputs. The detection of the bandwidth limitation and the selection of the appropriate specialization action (i.e., filtering or change in priority) should happen autonomically, without additional intervention of the simulation user.

5.2.4.1 Structured Data Transport: EVPath

After data events have been captured through the DataTap implementation of the ADIOS interface, an event processing architecture is provided in support of high-performance data streaming in networks with internal processing capacity. EVPath, the newest incarnation of a publish/subscribe infrastructure developed over many years [19, 20], is designed to allow for easy implementation of overlay networks with active data processing, routing, and management at all points within the overlay. In addition, EVPath allows the use of a higher-level control substrate to enable global overlay creation and management. Domain-specific control layers allow the management of the overlay to best utilize the underlying physical resources and provide for overlays that best address the application needs. For instance, the IFLOW management layer described in [11] is best suited for large-scale, wide-area, streaming applications, whereas another version of a control layer is more suitable for a massively parallel processor (MPP) such as the Cray system, where management components on compute nodes have limited ability for interaction with external control entities.

The basic building block in EVPath is a *stone*. An overlay *path* is comprised of a number of connected stones. A stone is a lightweight entity that roughly corresponds to processing points in a dataflow diagram. Stones can perform different types of data filtering and data transformation, as well as transmission of data between processes over network links.

EVPath is designed to support a flexible and dynamic computational environment where stones might be created on remote nodes and possibly relocate during the course of the computation. In order to support such an environment, we use a sandboxed version of C, coupled with a dynamic code-generation facility to allow native binary transformation functions to be deployed anywhere in the system at runtime [21]. The interface allows for the specification of data gateways (pass/no-pass) and data transformations (sum aggregation trees), and calls out to more specialized code (for example, invocation of a signed, shared library for performing FFTs). From these elements, the application user can specify in much greater detail how the interaction between the output of the running code and the data stored for later use should look.

5.2.4.2 Data Workspaces and Augmentation of Storage Services

As a concrete example of the user-driven interfaces that can be provided for application scientists, it is useful to consider the concept of data workspaces. In a data workspace, users are provided with an execution model (i.e., a semitransparent way of creating and submitting batch MPI jobs), along with a way for specifying the data control networks for how this data should move and be interpreted while in transit from the computing resource to the storage. Note that this concept interacts cleanly with the concept of a workflow — it is a part of a rich transport specification that then feeds the manipulation of the data once it has reached disk.

As an example of this concept, a team at Georgia Institute of Technology has built a data workspace for molecular dynamics applications that can make synchronous tests of the quality of the data and use that to modify the priority and even the desirability of moving that data into the next stage of its workflow pipeline [14]. Specifically, this workspace example modifies a storage service (ADIOS) that the molecular dynamics program invokes. As an example scenario, consider an application scientist who runs the parallel data output through an aggregation tree so that there is a single unified dataset (rather than a set of partially overlapping atomic descriptors), and then undergoes data quality and timeliness evaluation. Raw atomic coordinate data is compared to a previous graph of nearest neighbors through the evaluation of a central symmetry function to determine if any dislocations (seed of crack formation) have occurred in the simulated dataset. The frequency of the data storage is then changed, in this particular case, dependent on whether the data is from before, during, or after the formation of a crack, since the data during the crack formation itself is of the highest scientific value.

Similarly, in [14], data quality can be adapted based on a requirement for timeliness of data delivery — if a particular piece of data is too large to be delivered within the deadline, user-defined functions can be chosen autonomically to change data quality so as to satisfy the delivery timeline. In the case of an in-line visualization annotation, one could consider deploying a host of visualization-related functions — changing color depth, changing frame rate, changing resolution, visualization-specific compression techniques, and so forth Based on the user-specified priorities (color is unimportant, but frame rate is crucial), the in-transit manipulation of the extracted data allows for a much higher fidelity interaction for the application scientist. As the adaptation of the data stream becomes more complex, it leads naturally to discussion of full-fledged autonomic control of the network and computer platforms, which is the topic of the next sections.

5.2.4.3 Autonomic Data Movement Services Using IQ-Paths

Among data-driven high-performance applications, such as data mining and remote visualization, the ability to provide quality of service (QoS) guarantees is a common characteristic. However, due to most networking infrastructure being a shared resource, there is a need for middleware to assist end-user applications in best utilizing available network resources.

An IQ-Path is a novel mechanism that enhances and complements existing adaptive data streaming techniques. First, IQ-Paths dynamically measure [22, 23] and also predict the available bandwidth profiles on the network links. Second, they extend such online monitoring and prediction to the multilink paths in the overlay networks used by modern applications and middleware. Third, they offer automated methods for moving data traffic across overlay paths, including splitting a data stream across multiple paths and dynamically differentiating the volume and type of data traffic on each path. Finally IQ-Paths use statistical methods to capture the noisy nature of available network bandwidth, allowing a better mapping to the underlying best-effort network infrastructure.

The overlay implemented by IQ-Paths has multiple layers of abstraction. First, its *middleware underlay* — a middleware extension of the network underlay proposed in [24] — implements the execution layer for overlay services. The underlay is comprised of processes running on the machines available to IQ-paths, connected by logical links and/or via intermediate processes acting as router nodes. Second, underlay nodes continually assess the qualities of their logical links as well as the available resources of the machines on which they reside. Figure 5.4 illustrates an overlay node part of an IQ-Path. The routing and scheduling of application data is performed with consideration of path information generated by the monitoring entities. The service guarantees provided to applications are based on such dynamic resource measurements, runtime admission control, resource mapping, and a self-regulating packet routing and scheduling algorithm. This algorithm, termed PGOS (predictive guarantee overlay scheduling), provides probabilistic guarantees for the

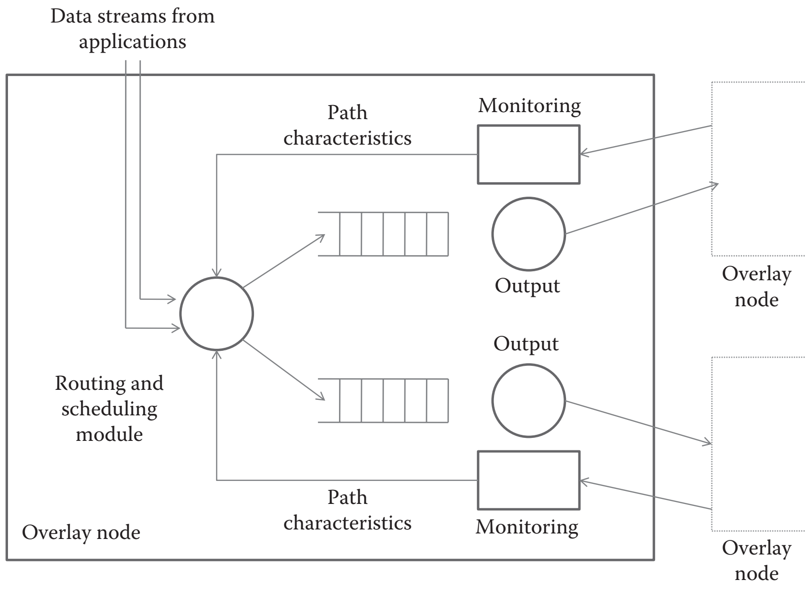


Figure 5.4 Data Movement services using IQ-Path.

available bandwidth, packet loss rate, and round-trip time (RTT) attainable across the best-effort network links in the underlay. More information on IQ-Paths can be found at [25].

IQ-Paths and PGOS provide the following services:

- **Probabilistic and "violation bound" guarantees.** Using the PGOS algorithm, service guarantees can be provided using network behavior prediction. PGOS can ensure that applications receive the bandwidths they require with high levels of accuracy (e.g., an application receives its required bandwidth 99% of the time), and that the occurrence of any violations, such as missed packet deadlines, is bound (e.g., only 0.1% of packets miss their deadline).
- **Reduced jitter.** Buffering requirements are minimized by reducing jitter in time-sensitive applications.
- **Differentiated streaming services.** Higher priority streams receive better service when network approaches maximum utilization.
- **Full bandwidth utilization.** Even with guarantees, the available and utilized bandwidths are not sacrificed.

5.3 Autonomic Services for Wide-Area and In-Transit Data

Complementary of the low-overhead asynchronous data extraction capabilities provided by ADIOS and its underlying mechanisms (i.e., DataTap and DART), wide-area streaming aims at efficiently and robustly transporting the extracted data from live simulations to remote services. In the previous sections we talked about services that worked on the local area network, and in this section we discuss services that must work over the wide area network. For example, in the context of the DOE SciDAC CPES fusion simulation project [26], a typical workflow consists 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 to remote sites and transformed along the way, for online simulation monitoring and control, simulation coupling, data analysis and visualization, online validation, and archiving. Wide-area data streaming and in-transit processing for such a workflow must satisfy 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 quality of service (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

Please provide full term.

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. (5) Effectively manage in-transit processing while satisfying the above requirements. This is particularly challenging due to the heterogeneous capabilities and dynamic capacities of the in-transit processing nodes.

5.3.1 An Infrastructure for Autonomic Data Streaming

The data streaming service described in this section is constructed using the Accord programming infrastructure [27–29], which provides the core models and mechanisms for realizing self-managing Grid services. These include autonomic management using rules as well as model-based online control. Accord extends the service-based Grid programming paradigm to relax static (defined at the time of instantiation) application requirements and system/application behaviors and allows them to be dynamically specified using high-level rules. Further, it enables the behaviors of services and applications to be sensitive to the dynamic state of the system and the changing requirements of the application, and to adapt to these changes at runtime. This is achieved by extending Grid services to include the specifications of policies (in the form of high-level rules) and mechanisms for self-management, and providing a decentralized runtime infrastructure for consistently and efficiently enforcing these policies to enable autonomic self-managing functional interaction and composition behaviors based on current requirements, state, and execution context.

An autonomic service extends a Grid service (such as the in-transit services described above) with a control port, as shown in Figure 5.5. The control augments the functional and operational ports that typically define computational elements, and supports external monitoring and steering. An autonomic service also encapsulates a service manager, shown in Figure 5.5 on the right, which monitors and controls the runtime behaviors of the managed service according to changing requirements and state of applications as well as their execution environment based on user-defined rules. As shown in the figure, the manager uses the local state of the element as well as its context along with user-defined rules to generate adaptations as well as management events. The control port (Figure 5.5, left) consists of sensors that enable the

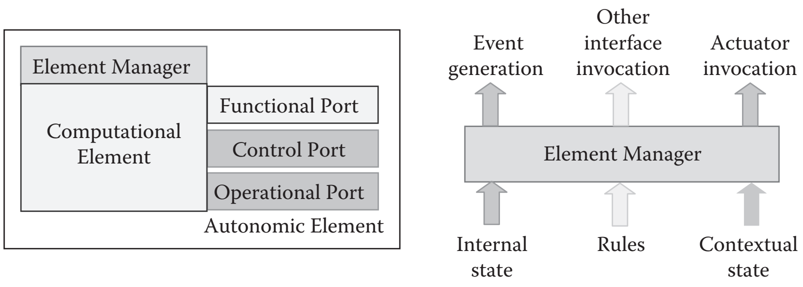


Figure 5.5 An autonomic service in Accord.

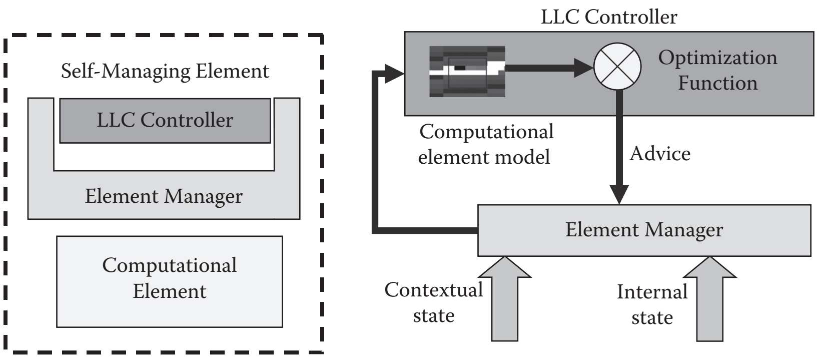


Figure 5.6 Self-management using model-based control in Accord.

state of the service to be queried, and actuators that enable the behaviors of the service to be modified. Rules are simple if-condition-then-action statements described using XML and include service adaptation and service interaction rules. Accord is part of Project AutoMate [29], which provides the required middleware services.

The element (service) managers within the Accord programming system are augmented with online controllers [30, 31] as shown in Figure 5.6. The figure shows the complementary relationship of an element manager and the limited look-ahead controller (LLC) within an autonomic element. 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 controllers decide when and how to adapt the application behavior, and the managers focus on enforcing these adaptations in a consistent and efficient manner.

We use the Accord programming system described above to address end-to-end QoS management and control at two levels shown in Figure 5.7. The first level in this figure is at the end points using adaptive buffer management mechanisms and proactive QoS management strategies based on online control and user-defined polices [30–32]. The second level shown in the figure is at the in-transit processing nodes, which are resources in the data path between the source and the destination, using reactive runtime management strategies, adaptive buffer management mechanisms [33, 34]. These two levels of management operate cooperatively to address overall application constraints and QoS requirements.

QoS management at application end-points The QoS management strategy at the application end-points combines model-based LLCs and policy-based managers with adaptive multithreaded buffer management [35]. The application-level data streaming service consists of a service manager and an LLC controller. The QoS manager monitors state and execution context, collects and reports runtime information, and enforces adaptation actions determined by its controller. Specifically, the controller

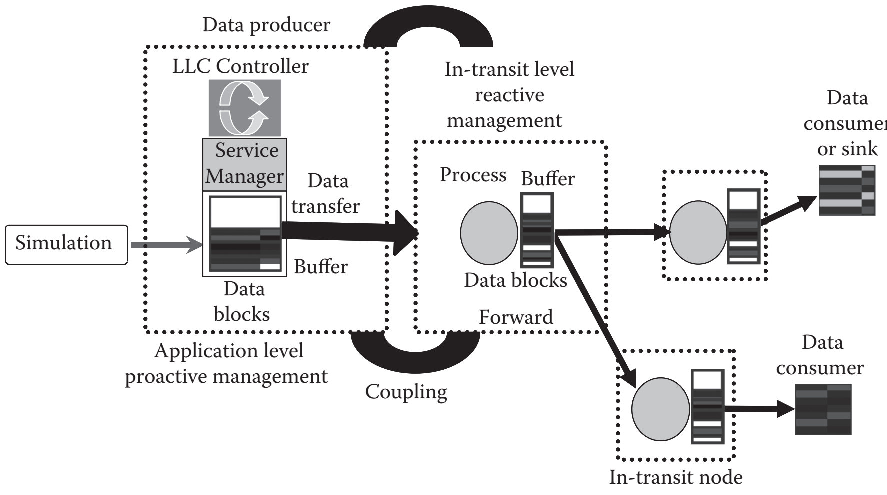


Figure 5.7 Overview of the two-level cooperative QoS management strategy.

decides when and how to adapt the application behavior, and the QoS manager focuses on enforcing these adaptations in a consistent and efficient manner. The effectiveness of this strategy was experimentally demonstrated in [32], which showed that it reduced overheads on the simulation (less than 5%) as well as buffer overflow and data loss.

5.3.2 QoS Management at In-Transit Nodes

In-transit data processing is achieved using a dynamic overlay of available nodes (workstations or small to medium clusters, etc.) with heterogeneous capabilities and loads — note that these nodes may be shared across multiple applications flows. The goal of in-transit processing is to opportunistically process as much data as possible before the data reaches the sink. The in-transit data processing service at each node performs three tasks, namely, processing, buffering and forwarding, and the processing depends on the capacity and capability of the node and the amount of processing that is still required for a data block at hand. The basic idea is that the in-transit data processing service at 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. Key aspects of the in-transit QoS management include: (1) adaptive buffering and data streaming that dynamically adjusts buffer input and buffer drainage rates, (2) adaptive run-time management in response to network congestions by dynamically monitoring the utility and tradeoffs of local computation versus data transmission, and (3) signal the application end-points about network state to achieve cooperative end-to-end self-management — that is, the in-transit management reacts to local services while the application end-point management responds more intelligently by adjusting its controller parameters to alleviate these congestions.

Experiments were conducted using the cooperative end-to-end self-managing data streaming using the GTC fusion application [32, 33] have shown that adaptive processing by the in-transit data processing service during congestions decrease the average percent idle time per data block from 25% to 1%. Furthermore, coupling end-point and in-transit level management during congestion reduces percent average buffer occupancy at in-transit nodes from 80% to 60.8%. Higher buffer occupancies at the in-transit lead to failures and result in in-transit data being dropped, and can impact the QoS of applications at the sink. Finally end-to-end cooperative management decreases the amount of data lost due to congestions at intermediate in-transit nodes, increasing the QoS at the sink. For example, if the average processing time per data block (1 block is 1 MB) is 1.6 sec at the sink, cooperative management saves about 168 sec (approx. 3 minutes) of processing time at the sink.

5.4 Conclusions

As the complexity and scale of current scientific and engineering applications grow, managing and transporting the large amounts of data they generate is quickly becoming a significant bottleneck. The increasing application runtimes and the high cost of high-performance computing resources make online data extraction and analysis a key requirement in addition to traditional data I/O and archiving. To be effective, online data extraction and transfer should impose minimal additional synchronization requirements, should have minimal impact on the computational performance, maintain overall QoS and ensure that no data is lost.

A key challenge that must be overcome is getting the large amounts of data being generated by these applications off the compute nodes at runtime and over to service nodes or another system for code coupling, online monitoring, analysis, or archiving. To be effective, such an online data extraction and transfer service must (1) have minimal impact on the execution of the simulations in terms of performance overhead or synchronization requirements, (2) satisfy stringent application/user space, time, and QoS constraints, and (3) ensure that no data is lost. On most expensive HPC resources, the large numbers of compute nodes are typically serviced by a smaller number of service nodes where they can offload expensive I/O operations. As the result, the I/O substrate should be able to asynchronously transfer data from compute nodes to a service node with minimal delay and overhead on the simulation. Technologies such as RDMA allow fast memory access into the address space of an application without interrupting the computational process, and provide a mechanism that can support these requirements.

In this chapter we described the ADIOS I/O system and underlying mechanisms, which represent a paradigm shift in which I/O in high-performance scientific application is formulated, specified, and executed. In this new paradigm, the construction of the writes and reads within the application code is decoupled from the specification of how that I/O should occur at runtime. This allows the end user substantial additional flexibility in making use of the latest in high-throughput and asynchronous I/O methods without rewriting (or even relinking) their code. The underlying mechanisms include low-level interfaces which enable lightweight data capture, asynchronous data movement, and specialized adaptive transport services for MPP and wide-area systems. Our experiences with a number of fusion and other codes have demonstrated the effectiveness, efficiency, and flexibility of the ADIOS approach and the accompanying technologies such as DataTaps, I/O graphs, DART, and the autonomic data management, transport, and processing services. These services use metadata that effect I/O operations and access of parallel file systems. Other aspects of metadata are discussed in Chapter 12.

References

For all websites, please include date accessed if known.

[1] D3D, D3D https://fusion.gat.com/global/DIII-D, 2008.

[2] M. Ono et al., Exploration of sperical torus physics in the NSTX device, *Nuclear Fusion*, vol. 40, pp. 557–561, 2000.

[3] ITER, ITER http://www.iter.org, 2008.

[4] S. E. Z. Lin, T. S. Hahm, and W. M. Tang, size scaling of turbulent transport in magnetically confied plasmas, *Phys. Rev. Letters*, vol. 88, 2002.

[5] C.-S. C. S. Ku, M. Adams, F. Hinton, D. Keyes, S. Klasky, W. Lee, Z. Lin, and S. Parker, Gyrokinetic particle simulation of neoclassical transport in the pedestal/scrape-off region of a tokamak plasma, *Institute of Physics Publishing Journal of Physics: Conference Series*, pp. 87–91, 2006.

[6] M. W. Hasan Abbasi, and K. Schwan, LIVE data workspace: A flexible, dynamic and extensible platform for petascale applicatons, in *IEEE Cluster 2007*, 2007.

[7] C. Docan, M. Parashar, and S. Klasky, High speed asynchronous data transfers on the Cray XT3, *Cray User Group Conference*, 2007–05 2007.

[8] L. L. N. Laboratory, VisIt Visualization Tool: https://wci.llnl.gov/codes/visit/.

[9] O. Messer, S. Bruenn, et al., Petascale supernove simulation with CHIMERA, *Journal of Physics Conference Series*, vol. 78, 2007.

Please include page numbers.

[10] G. Eisenhauer, F. Bustamante, and K. Schwan, A middleware toolkit for client-initiated service specialization, in *PODC Middleware Symposium*, 2000.

[11] F. E. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, Efficient wire formats for high performance computing, in *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM)*, 2000, p. 39.

[12] G. Eisenhauer, F. E. Bustamante, and K. Schwan, Event services for high performance computing, in *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, 2000, p. 113.

Please include publisher information

[13] G. Eisenhauer, PBIO http://www.cc.gatech.edu/systems/projects/PBIO.

[14] M. Wolf, H. Abbasi, B. Collins, D. Spain, and K. Schwan, Service augmentation for high end interactive data services, in *IEEE Cluster Computing Conference (Cluster '05)*, Boston, MA, 2005.

[15] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan, Resource-aware distributed stream management using dynamic overlays, in *the 25th IEEE International Conference on Distributed Computing Systems (ICDCS-2005)*, Columbus, OH, 2005.

[16] P. W. Ron Oldfield, A. Maccabe, Lee Ward, and T. Kordenbrock, Efficient data-movement for lightweight I/O, in *The 2006 IEEE International Conference on Cluster Computing (Cluster 2006)*, Barcelona, 2006.

[17] M. Wolf, Z. Cai, W. Huang, and K. Schwan, SmartPointers: Personalized scientific data portals in your hand, in *Proceedings of the Proceedings of the IEEE/ACM SC2002 Conference*, 2002, p. 20.

[18] G. Eisenhauer, Portable self-describing binary data streams, 1994.

[19] G. Eisenhauer, The ECho Event Delivery System (GIT-CC-99-08), in *Georgia Tech College of Computing Technical Reports* (ftp://ftp.cc.gatech.edu/pub/coc/tech_reports), 1999.

[20] G. Eisenhauer, F. Bustamante, and K. Schwan, Native data representation: An efficient wire format for high-performance distributed computing, *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 1234–1246, Dec 2002.

[21] B. Plale, G. Eisenhauer, L. K. Daley, P. Widener, and K. Schwan, Fast heterogeneous binary data interchange for event-based monitoring, in *International Conference on Parallel and Distributed Computing Systems (PDCS2000)*, 2000.

[22] M. Jain and C. Dovrolis, End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput, *Proceedings of the 2002 SIGCOMM conference*, vol. 32, pp. 295–308, 2002.

[23] M. Jain and C. Dovrolis, Pathload: A measurement tool for end-to-end available bandwidth, *Passive and Active Measurements*, vol. 11, pp. 14–25, 2002.

[24] N. Akihiro, P. Larry, and B. Andy, A routing underlay for overlay networks, in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany: ACM, 2003.

[25] Z. Cai, V. Kumar, and K. Schwan, IQ-Paths: Self-regulating data streams across network overlays, 2006.

Please include publisher information

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

[27] H. Liu, V. Bhat, M. Parashar, and S. Klasky, An autonomic service architecture for self-managing grid applications, in *6th International Workshop on Grid Computing (Grid 2005)*, Seattle, WA, USA, 2005, pp. 132–139.

[28] H. Liu and M. Parashar, Accord: A Programming framework for autonomic applications, *IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Engineering Autonomic Systems*, vol. 36, pp. 341–352, 2006.

[29] M. Parashar, H. Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang, and S. Hariri, AutoMate: Enabling autonomic applications on the grid, *Cluster Computing* vol. 9, pp. 161–174, April 2006.

[30] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, and S. Abdelwahed, Enabling self-managing applications using model-based online control strategies, in *3rd IEEE International Conference on Autonomic Computing*, Dublin, Ireland, 2006, pp. 15–24.

[31] 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, 2006, pp. 176–183.

[32] V. Bhat, M. Parashar, and N. Kandasamy, Autonomic data streaming for high performance scientific applications, in *Autonomic Computing: Concepts, Infrastructure and Applications*, M. Parashar and S. Hariri, Eds.: CRC Press, 2006, pp. 413–433.

[33] V. Bhat, M. Parashar, and S. Klasky, Experiments with in-transit processing for data intensive grid workflows, in *8th IEEE International Conference on Grid Computing (Grid 2007)*, 2007.

[34] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, S. Klasky, and S. Abdelwahed, An self-managing wide-area data streaming service, *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, vol. 10, pp. 365–383, December 2007.

[35] V. Bhat, S. Klasky, S. Atchley, M. Beck, D. McCune, and M. Parashar, High performance threaded data streaming for large scale simulations, in *5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, PA, 2004, pp. 243–250.