Flexpath: Type-Based Publish/Subscribe System for Large-scale Science Analytics

Jai Dayal, Drew Bratcher,
Greg Eisenhauer, Karsten Schwan,
Matthew Wolf, Xuechen Zhang
Georgia Institute of Technology
Atlanta, GA
{*jdayal3, dbratcher*}*@gatech.edu*
{*eisen, karsten.schwan, mwolf, xczhang*}*@cc.gatech.edu*

Hasan Abbasi, Scott Klasky,
Norbert Podhorszki
Oak Ridge National Labs
Oak Ridge, TN
{*habbasi, klasky, pnorbert*}*@ornl.gov*

***Abstract*—As high-end systems move toward exascale sizes, a new model of scientific inquiry being developed is one in which online data analytics run concurrently with the high end simulations producing data outputs. Goals are to gain rapid insights into the ongoing scientific processes, assess their scientific validity, and/or initiate corrective or supplementary actions by launching additional computations when needed. The *Flexpath* system presented in this paper addresses the fundamental problem of how to structure and efficiently implement the communications between high end simulations and concurrently running online data analytics, the latter comprised of componentized dynamic services and service pipelines.**

Using a type-based publish/subscribe approach, Flexpath encourages diversity by permitting analytics services to differ in their computational and scaling characteristics and even in their internal execution models. Flexpath uses direct and MxN connections between interacting services to reduce data movements, to allow for runtime connectivity changes to accommodate component arrivals/departures, and to support the multiple underlying communication protocols used for analytics workflows in which simulation outputs are processed by analytics services residing on the same nodes where they are generated, on the same machine, and/or on attached or remote analytics engines. This paper describes the design and implementation of Flexpath, and evaluates it with two widely used scientific applications and their associated data analytics methods.

***Keywords*-Publish/Subscribe, Data Staging, Data Analytics, in-Situ, Scalable I/O, Code Coupling**

I. Introduction

The push towards exascale is causing researchers to rethink the way scientific applications and systems will operate and interact. The consensus is that at such large scales, applications can no longer be structured as single bulk-synchronous codes, but will instead be comprised of sets of heterogeneous parallel components that interact via complex runtime data exchanges [1], [2]. Our research is concerned with one specific class of such applications, akin to a "Computational Wet Lab", in which large-scale parallel codes simulating some scientific process are the core engines driving multiple, concurrently running, dynamically organized sets of analysis and data processing codes that inspect and evaluate simulation outputs. Typical online analytics tasks performed by these codes are to visualize output data [3], [4], prepare data for efficient storage and retrieval [5], [6], ascertain the simulation's validity, and/or take corrective actions to change the simulation's execution [7].

Important for the emerging online analytics model for exascale applications is the efficient and flexible data exchange between heterogeneous parallel codes. For example, in the S3D combustion model code [8], the simulation's raw output data is fed into sets of services structured as analytics workflows. Such services may run on the same nodes as those used by the simulation, on staging areas located on the same machine, on attached analytics engines, or at remote locations as part of enabling multi-institution collaborations.

In this simulation/analysis model for scientific investigation, rapid inquiries into the ongoing scientific processes being simulated are a key requirement. As such, analytics services may need to adapt to dynamic simulation behavior at runtime, an example being the simulation's use of runtime mesh refinement which gives rise to changes in the service's input & output volumes and patterns. They may also need to react to runtime changes in science user needs, as when new analytics are launched in response to interesting scientific events. Further, with scientists routinely using a plethora of useful analytics and visualization tools and systems [9], [4], the analytics service environment may reflect highly diverse implementations and requirements, including various models of parallelism, different algorithmic constructs like domain decompositions or octrees, and as indicated above, may run across several types of hardware at multiple physical locations.

This paper presents *Flexpath*, a *type-based* publish/subscribe infrastructure for coupling high-end scientific applications with their online analytics services. Flexpath's pub/sub approach makes possible runtime configurability, scalability, and also fault tolerance, as the pub/sub abstraction allows for the decoupling of diverse analytics components, permits multiple subscribers or publishers to share a single data stream, and suppresses communications for cases in which there are no subscribers to certain data streams

(e.g., those not of current interest). This is particularly well suited for the "Computational Wet Lab" approach, as the core simulation may therefore be structured to make available a substantial array of internal data, knowing that only those parts that are needed at runtime will actually be exported. These properties contrast with the typical assumptions made by communication infrastructures like MPI, where the domain of executing processes is initialized at launch and cannot grow or shrink for the remainder of the execution.

With Flexpath, one can construct and dynamically manage or change the data processing pipelines or workflows needed for runtime analysis of the large volumes of this output data in ways that meet the following four design requirements of these sorts of applications: (1) decouple analytics services from simulation codes, (2) maintain levels of performance similar to those obtained by analytics routines statically embedded with simulations, (3) permit those pipelines to cross node and/or machine boundaries, and (4) support the creation of higher level methods for managing these pipelines. Sample management constructs built in our own previous work [10], for example, have balanced pipeline operations to ensure QoS and have implemented transactional constructs with the goal of providing ACID properties for select online analytics [11], [12].

Flexpath's pub/sub communication mechanism, key to meeting design objective (1), obtains flexibility for component-component communications, without the performance penalties incurred by traditional broker-based pub/sub infrastructures. This technical contribution is achieved by using direct connections between interacting components, including the scatter-gather or MxN communications needed across different communicating internally parallelized analytics components. This high performance implementation for such peer-to-peer techniques utilizes a subscription implementation, allowing readers to specify derived versions of messages, e.g., to receive only those slices of data objects they require, as well as registering dynamic transformations of typed objects when there are mismatches between publishers and subscribers, e.g. row to column order array conversions.

With regards to the need to maintain performance in cross-platform environments (design objectives 2 and 3), Flexpath has been built to leverage multiple underlying communication protocols, ranging from a shared memory protocol employed for on-node communications, to the RDMA-based protocols existing on high end machines, to the TCP/IP protocols required for linking remote collaborators. As is described in Section IV, much of this comes from inheriting a multi-modal connection management system through the EVPath framework. Finally, with regards to design object (4)'s concerns for management, Flexpath's approach allows for it to export monitoring data and management 'hooks' with which higher level management methods can be realized. As will be seen later, we utilize some simple workflow-level management schemas in this work, but future work will extend the complexity and robustness of this feature of the system.

Conceptually, Flexpath's development builds on extensive prior work on efficient parallel I/O pipelines, including data staging methods for running analytics and visualization [13], [5], [14], [15], data streaming and the online QoS control of such data streams [16], [17], [11], the aggressive use of source-based data reduction and filtering [6], [5], and convenient ways to carry out remote data visualization [18], [9], [4]. For high end machines, challenges include dealing with network congestion [13], providing data reliability when operating at scale [12], making data 'right' for use by successive analytics codes without unnecessary data movement [19], [5], and dealing with application dynamics, as when codes are dynamically activated or de-activated. Such dynamics, in fact, have given rise to interesting methods used by modern data visualization systems like VisIt contracts [20].

Driven by such prior work, Flexpath is designed as a communication substrate that does not proscribe specific management methods. Instead, it makes possible the efficient realization of alternative communication scheduling techniques [13] and/or higher level methods for managing entire analytics workflows [10]. In contrast to web-based or commercial data streaming infrastructures [17], [21], [22], it does not constrain end users in how to write their analytics routines, so that they can leverage the rich tools already existing for these purposes, like R or MatLab. Finally, leveraging the ADIOS I/O APIs already in common use on petascale machines [23], Flexpath's implementation as an ADIOS 'transport method' allows it to adopt and adapt many off-line analytics pipelines that were originally structured as sets of independently programmable and deployable analytics services using ADIOS as the interface of choice [23].

Flexpath is deployed for use across a range of high end machines, including ORNL's Titan machine, Infiniband clusters, and commodity scientific computing engines. This paper experimentally evaluates its technical elements and approach with two representative applications with significant scientific user communities, LAMMPS [24] and GTS [25], coupled with their associated data analytics service flows. For this paper, experiments are run on Oak Ridge National Lab's Sith machine and on smaller-scale Linux clusters available at our own institution.

The remainder of this paper is organized as follows. Section II describes two scientific applications guiding our research. Section III outlines *type-based* publish/subscribe and describes how such abstractions have to be augmented to be made suitable for high end applications. Section IV describes the design and implementation of Flexpath. Section V presents experimental results, and Section VI overviews related work. Conclusions and future work appear

Table I
CHARACTERISTICS OF SMARTPOINTER ANALYSIS ACTIONS

	Complexity	Data Model	Stateful
Helper	$O(n)$	Array	No
Bonds	$O(n^2)$	Array, Parallel	No
CSym	$O(n)$	Complex	Yes
CNA	$O(n^3)$	Array	No

in Section VII.

II. APPLICATION DRIVERS

As an exemplar of the type of analytically-rich scientific pipelines mentioned above, we offer a scenario based on understanding crack genesis and propagation in advanced materials. At its core, this work uses LAMMPS (Large Scale Atomic/Molecular Massively Parallel Simulator) [24], which is a molecular dynamics simulation workhorse used across a number of scientific domains, including materials science, biology, and physics. It is written with MPI and performs force and energy calculations on discrete atomic particles. After a number of user-defined epochs, it outputs the atomistic simulation data (positions, atom types, etc.), with the size of this data ranging fom megabytes to terabytes depending on the science being conducted.

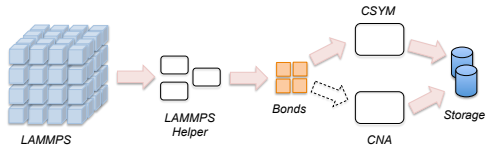


Figure 1. LAMMPS and SmartPointer Analysis Pipeline

SmartPointer is a representative analytics pipeline interpreting LAMMPS output data to detect and then scientifically explore plastic deformation and crack genesis. In such scenarios, the material being simulated is steadily stressed until it first starts to break. The scientific question being addressed by this research is how to understand the geometry of the region around that initial break, particularly when it is a multi-crystalline, nano-structured material. This means that the purpose of the molecular dynamics simulation is to bring the data set to some self-consistent, interesting state, at which point substantial additional analytics and characterization need to take place. The SmartPointer analytics toolkit implements these functions to determine where and when plastic deformation occurs and to generate relevant information as the material is cracked. Table I summarizes the computational characteristics of the individual SmartPointer components, and the list below explains each in greater detail.

- *Lammps Helper*: serves as an aggregator and filter of the raw LAMMPS data.
- *Bonds*: subscribes to aggregated data from Lammps helper at each time-step, and performs an all-nearest neighbor calculation to determine which atoms are bonded together in order to publish a bond-pair array as its output.
- *Csym*: the central symmetry analysis routine operates on an array of bond-pairs from bonds, and also a bond adjacency list (a graph structure) to determine if there is a deformation in the material. This code maintains the initial bonds adjacency state for the duration of its run.
- *CNA*: common neighbor analysis executes whenever CSYM determines that a deformation in the material has occurred. CNA is compute-intensive and is executed on the bond pairs array to perform a structural characterization on the data, to determine the conditions under which the crack occurred.

As an alternative application example, we also present GTS. GTS is a plasma fusion simulation that solves the gyrokinetic equation with an implementation that exploits coarse grained process level parallelism using MPI, and more fine-grained thread-level parallelism using OpenMP [25]. This particle-in-cell code has different output frequencies for both particles and mesh-level statistics. In order to examine the dynamics involved, in particular dangerous transient effects that might damage a real reactor vessel, it is useful to dynamically evaluate and characterize particular trends on the inner and outer edges of the plasma. Unlike the LAMMPS case, these transients are not as algorithmically identifiable, so secondary analysis methods are used to infer their existence, and then, much more detailed inspection involving direct interaction with the physicists is used to further the investigation. The GTS analytics pipeline used in this paper computes parallel histograms of multiple grid-carried variables, and runs parallel-coordinate visualizations to provide suitable data to those physicists.

The LAMMPS and GTS analytics workflows have some important shared characteristics. In both examples, analytics codes are run as independent services, each simply executing its functions on the data that is available. The operation of the individual analysis routines are not affected by each other and generally, the codes are unaware of the details of how or when the other codes are run. This results in analytics workflows best described as sets of analytics services loosely coupled in terms of space, time, and synchronization.

III. TYPE-BASED PUBLISH SUBSCRIBE FOR HPC ENVIRONMENTS

Type-based publish/subscribe [26] is a pub/sub paradigm in which producers publish objects classified by type to a communication substrate, and consumers subscribe to them by specifying the types of objects in which they are interested. Here, type reflects both the structure of the published data as well as metadata extensions that can be determined at runtime. This distinction, along with other technical contributions, is part of what allows Flexpath to adopt a high performance direct-connect, rather than brokered, infrastructure.

While Flexpath uses a type-based pub/sub model, end users are not required to change their ADIOS codes or

applications to adopt this new model. Instead, the Flexpath implementation exploits the model's several similarities with the standard file I/O model already known to science users. In the file I/O model, science applications exchange data by using a shared filesystem as the data exchange medium [27], and workflows are constructed through the use of intermediate files stored on disk. As a familiar scientific scenario, consider the following: at each output epoch the parallel writers open a file, encode their data in the proper metadata-rich serialization format, like NetCDF [28] or HDF5 [29], populate the write buffers, and finally, flush them to disk. Similarly, for a given read epoch the (possibly parallel) readers open the file, read metadata about the objects present in the file, create the appropriate buffers, perform the reads, and then seek ahead to the next block of data, if available. Additional attractive elements of the file-based approach include the ability to perform "seeks" to retrieve fine-grained slices of the available data, data durability and persistence guarantees.

A detraction from this model is the synchronous nature of file-based I/O and its poor performance at large scales. Particularly, if there is a complex trade-off between number of files, number of writers, and the layout of data within those files to be most scientifically useful, the attempt to optimize any one file system parameter can yield sub-optimal results for the other. In the dynamic scientific investigations targeted by this work, a mis-predicted optimization could have profound impact on the viability of the runtime analysis if it were to use a traditional filesystem-based approach. A key reason for introducing our asynchronous, type-based pub/sub has been to avoid this disk bottleneck when linking simulations with dynamic sets of analytics services[5], [14], [13].

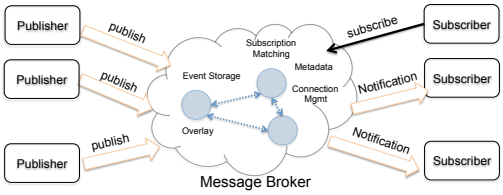
The properties of a type-based pub/sub system, although superficially quite different from file I/O, map relatively well to the subset of such general functions that are offered by high performance I/O libraries such as HDF5 or ADIOS. For example, file names serve as the naming convention for establishing a pub/sub "channel" between coupled applications, so that writers and readers can be logically mapped between publishers and subscribers to a shared data set. A key realization about the high performance I/O abstraction is that, since data is already laid out for the I/O system utilizing many higher-level concepts of data structure (arrays, slabs, meshes, etc.), a seek is not an arbitrary binary offset within the file. Instead, it maps quite well to metadata subscriptions or type-based derivations within the scope of type-based pub/sub. For example, a seek to a particular slice of a global array can also be interpreted by the pub/sub as a parameterization of the peer-to-peer subscription parameters. This structural, rather than byte-level, addressing of data in the high performance space is key to aligning the two paradigms.

Beyond this mapping of file I/O to equivalent pub/sub actions, there are additional properties of the online analytics workflows targeted by Flexpath that make them well suited for the type-based pub/sub paradigm. The input and output types of each component in the workflow are well defined, giving rise to clean mappings to equivalent pub/sub type descriptions. Additionally, since the components of the workflow operate independently of each other, this favors an asynchronous communication model not subject to the issues with tightly synchronized data exchanges in which senders block when downstream receivers are still processing the previous interval of data. This also supports the original design objective of being able to handle a heterogeneous computational environment. Finally, for analytics components that can change during the course of the run, key constituents of the types of workflows we aim to address, this amounts to dynamic changes in the data flow. Our pub/sub offers a model that allows such dynamics while not having to make the individual components aware of such run-time complexities.

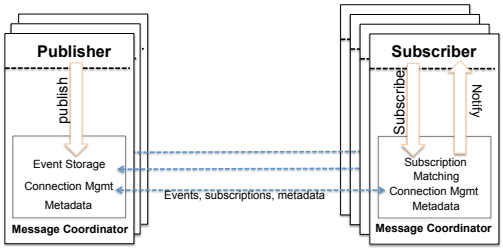
While conceptually attractive, the efficient implementation of type-based pub/sub for high end applications and platforms poses significant challenges. To obtain high performance for large data volumes, we cannot use overlay routing techniques and/or move data to third-party brokers, as done in other traditional pub/sub implementations [30], [31], [32]. Second, unlike traditional type-based pub/sub, a single data object represented by a type is a collection of messages matching this type obtained from some large number of sources producing these messages, i.e., each process in the parallel application produces a portion of a global array as well as scalar variables that describe both global knowledge and the process's local view of the global array. Thus, the definition of type has to be extended to include notions of both local and global metadata parameters. Third, in the common MxN data exchanges that occur among science codes, a subscriber only wants to receive certain slices of the objects or in fact, objects transformed from one type to another. So, in addition to specifying types, subscribers also need to specifiy derivations on types. As a final complication, a type-based pub/sub infrastructure must have ways of dealing with type augmentations at runtime in order to be useful for adaptive codes like S3D.

Naturally, there are also machine-specific challenges to an efficient implementation of pub/sub on large-scale supercomputers. Structuring a solution which can both efficiently utilize highly specialized networking hardware and protocols, such as Infiniband and Cray's Gemini interconnect and simultaneously operate across multiple networking technologies, e.g. TCP/IP, in order to extend workflows across multiple machines and geographies, requires great care. There are issues of placement, throughput matching, and even security that must be addressed while maintaining both high performance and the simple pub/sub abstraction.

In summary, scientific simulations with complex online



(a) Traditional Pub/Sub



(b) Flexpath Pub/Sub

Figure 2. Traditional model of publish/subscribe vs. Flexpath model of publish/subscribe allowing for fine-grained data exchanges across parallel applications.

analytics workflows can benefit from the pub/sub paradigm, but given the non-trivial data exchange and I/O characteristics of these applications, and the nature of the systems on which they run, type-based publish/subscribe must be rethought to enable data exchanges at scale. Connection management, type derivations, and subscription management all must be re-addressed. We next explain the Flexpath architecture and implementation as it addresses these challenges.

IV. Design and Implementation

Figure 2 depicts the conceptual design of Flexpath and contrasts it with the standard broker-based pub/sub model. The key design component allowing us to break away from the traditional broker model is the use of message coordinators local to each participant. In Flexpath, direct connections are established between coordinators on opposing sides; when joining a channel, a subscriber selects a publisher peer coordinator it uses to retrieve publisher metadata. This metadata is used in conjunction with subscriber subscriptions to establish connections with publishers that own the requested data. Additionally, control messages are sent across connected coordinators to perform coordinated control operations like the eviction of expired data from the local data stores. We next describe some of the background needed for understanding the implementation of this functionality.

A. Background Technologies

1) EVPath Overview: The Flexpath messaging infrastructure is built on the EVPath [16] event-based transport middleware. EVPath supports the construction of active messaging overlay networks. User-defined data filtering and transformation functions reside in lightweight "stones" that serve as processing points in the overlay, and stones are linked to form overlay "paths", where the processes hosting these stones may reside on the same physical machine, on cluster nodes, or even on machines at different geographical locations. The filtering and transformation functions run by stones are implemented by registered call-back handlers written in C and statically associated with stones, or as inline functions deployed at runtime generated with the CoD (C-on-Demand) language. The types of EVPath stones used in the Flexpath implementation are the following:

- *Terminal Stone*: runs an application-registered call-back handler associated with an event type; the handler is invoked upon receipt of such an event.
- *Multi-Queue Stone*: operates over a collection of typed events, and allows users to implement policies like a tumbling window policy, or perform event transformations that span multiple event types.
- *Bridge Stone*: is used for network transmission, for communication with stones in a remote address space.

The additional stone types present in EVPath are described in [16].

Flexpath adopts from EVPath its methods for data serialization, termed Fast Flexible Serialization (FFS) [33], which means that Flexpath events are comprised of typed data elements, with types seen by all of the stones (and functions) operating on those events. The basic types supported are similar to those present in the C language, but with FFS, those types can be the building block for event data comprised of complex graph structures. FFS serialization creates self-describing events, meaning that each event carries its typed data as well as sufficient metadata to identify those types. We note that functions coded with CoD manipulating FFS encoded events can be generated at runtime and dynamically deployed to stones, in contrast with handlers that are compiled and deployed statically. Finally, to operate across several diverse communication protocols, Flexpath uses EVPath's networking abstraction, termed Connection Manager(CM), which currently supports as lower level protocols TCP/IP sockets, and via Sandia's NNTI [34], also high performance protocols like Infiniband, Cray's Gemini, and the Bluegene interconnect.

2) ADIOS Interface: The Adaptable IO System (ADIOS) is an I/O componentization library that exposes file-like read and write interfaces to applications, with underlying I/O methods including disk based methods like POSIX and MPI-IO, and "staging" methods like Datatap[13], Dataspaces[14], and also Flexpath. With ADIOS, end users can simply 'switch' transports, without modifying their codes, using an external XML document identifying those transports as well as other I/O characteristics, like the variables to be written, their array dimensions & offsets, etc. We have chosen ADIOS to be the interface into Flexpath for two reasons: 1) For ease of use by the large number of existing applications that use the ADIOS interfaces, and 2) as describe in section III, there is a natural translation from ADIOS file-based I/O interfaces and type descriptions to Flexpath's pub/sub

approach.

B. Implementation

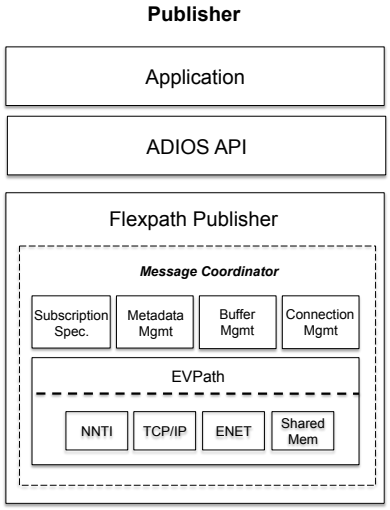


Figure 3. Software Architecture of Flexpath Publishers

Figure 3 depicts the software architecture of Flexpath from the publisher's perspective. The subscriber's interface is similar, except that it is layered beneath the ADIOS read interface.

Type Representation: The publisher side of Flexpath obtains type information about data from the ADIOS data descriptor, generated by parsing the XML document during the *adios_open* call. This information is converted into a FFS format header uniquely identified by a "cookie" transmitted along with the data. Upon the arrival of an event, if a receiver has not yet seen this cookie, the receiver issues a fetch request to the sender to obtain the FFS descriptor. This scheme avoids the redundant transmission of FFS metadata.

Publishing Data: Publishers submit their data in Flexpath through the *adios_write* call, which is called for each variable to be written. Flexpath copies the data into the appropriate location in the FFS encoded buffer. This extra copy is not inherent to the pub/sub model, but is performed to satisfy the safety requirement of the ADIOS interface, which allows user codes to manage their own buffers. At the end of the output epoch, publishers perform a *publish* operation, available through the *adios_close* call, which submits the FFS encoded data to the local message coordinator. Additionally, on the publish operation, if there are any global arrays, we distribute each publisher's array offset metadata to all other publishers, so that the subscriber can ask its peer publisher coordinator for this information directly. This metadata allows us to extend the traditional definition of types to include local pieces of a larger global object.

Subscriptions: Subscriptions are realized in three steps. First, the subscriber informs its local message coordinator about what variables and slices it needs. The message coordinator then fetches the global offset information for the given epoch from its peer writer coordinator and uses this information, if available, to determine from which publishers data is needed. The subscriber message coordinator will then send to each of those publisher coordinators a fetch message requesting the desired variables and slices. The offset metadata exchange also serves as our *notify* abstraction; metadata is only present for an output epoch if data for this epoch has been published.

In addition to the array slicing style subscriptions, we also allow for subscribers to specify type transformations, to allow publishers and subscribers to resolve type mismatches. In the example listed in Section II, the CSYM code actually wants to receive some data in the form of a bonds-adjacency list, a more complex graph structure, rather than only the bond-pair integer array published by the Bonds code. With Flexpath, this is done via transform operators, represented as CoD code or as a registered transform function. For this example, the transform function is run on the subscriber side to avoid having to transmit both sets of data.

Message Coordinators: Message Coordinators are implemented by EVPath stones, CoD code, and with callback handlers. On the publisher side, an EVPath multi-queue stone executes in the same thread as the one handling network communication, i.e., serving as an entry point for incoming messages and as the dispatcher for outgoing messages. To reduce its processing load, messages are forwarded to a control thread that handles message processing and any state changes within the Flexpath publishers. Each publisher message coordinator also maintains a local in-memory data buffer for storing published data as well as the associated metadata. It is this local data store, the control thread, and the communication thread that jointly realize Flexpath's asynchronous communication model. Additional functionality in message coordinators maintains reference counts to understand when data has been successfully received by all subscribers and perform subsequent data eviction operations, etc.

The subscriber's message coordinator is similar, except that it uses a terminal stone and call back handlers to invoke necessary state changes, and that received data is copied into the user's receive buffer registered through the ADIOS read interface. Finally, there is additional information about inter-coordinator connectivity, elided here for brevity.

V. Experimental Evaluation

Flexpath is evaluated experimentally using the Sith cluster hosted at Oak Ridge National Labs, and on the Windu and Jedi clusters hosted at Georgia Tech. The Sith machine is a 40 node cluster and each node is equipped with four 2.3 GHz 8 core AMD Opteron processors and 64 GB of memory. The system offers QDR Infiniband for Lustre and MPI traffic, and a 1Gb Ethernet link for communication across MPI domains.

The Windu and Vogue clusters operate as separate Infiniband domains and the two clusters share a 1Gb Ethernet

Table II
LAMMPS PIPELINE EXPERIMENTAL SETUP.

Data Size	LAMMPS	Helper	Bonds
76 MB	128	1	2
153 MB	256	2	4
305 MB	512	4	8
610 MB	1024	8	16

Table III
DATA SIZES AND CORE COUNTS FOR WEAK SCALING EXPERIMENTS

link for communication between the two. The nodes in both clusters contain one 2.67Ghz Intel Xeon 12 core processor and 48Gb ram.

A. *Affect on Application Execution Time*

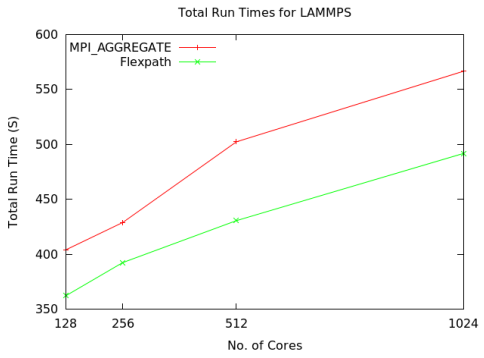
This experiment measures the effect on application level performance when constructing a workflow, using Flexpath as the data exchange mechanism. We measure time spent on output operations for each component in the pipeline and compare Flexpath's performance with that of the MPI_Aggregate synchronous disk-based method offered by the ADIOS interface. The MPI_Aggregate method is optimized for parallel Lustre I/O, and discussion on these optimizations is made available in [23].

We use weak scaling to show how the system behaves both in terms of larger numbers of participants and larger data volumes. Table II shows the data sizes LAMMPS produces at each output epoch, and the number of cores on which each code is executed. CSYM and CNA are serial codes that always run with an MPI size of 1, so we exclude them from the table.

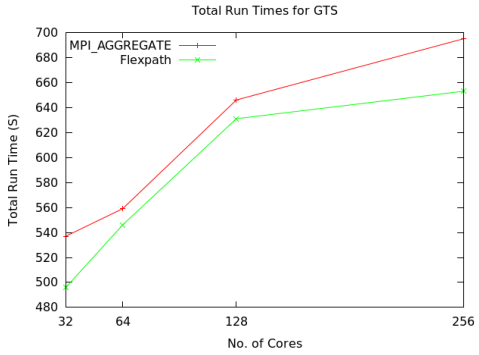
Figure 4 shows the total time each component in the LAMMPS analytics pipeline spends on performing I/O over its full execution. The LAMMPS application experiences a significant decrease in I/O time; when running on 1024 cores, it spends just over 1.3 seconds on I/O when using Flexpath vs. 117 seconds when using the MPI_Aggregate method. This is directly due to Flexpath's asynchronous nature.

Asynchronous operation also engenders reductions in I/O time for the other components in the pipeline, but the decrease is not as drastic, for several reasons: (1) the analytics components run at smaller MPI sizes than the LAMMPS application, so for each component process, Flexpath has a much larger volume of data to process and move; and (2) the Bonds and CNA components run slower than the others, so occasionally, there will be blocks in the analytics portion of the pipeline as the coordinator data stores become full. It is blocking issues in scenarios like these that motivate the notion of I/O Containers for managing analytics pipelines presented in [10].

Figure 5 depicts the overall improvements in runtimes for the LAMMPS and GTS applications. The decrease in the time spent on I/O translates to decreased run times. We note that there is some disparity between the reduced time spent on I/O and the total reductions seen in run-



(a) LAMMPS



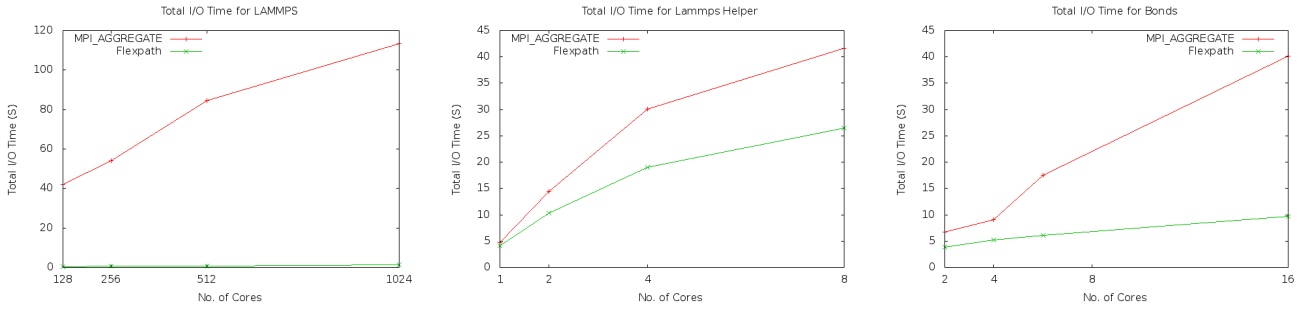
(b) GTS

Figure 5. Total Execution Times for LAMMPS and GTS applications.

times. This is because (i) Flexpath is an active I/O transport, so it will continue to operate and borrow CPU cycles during the application's normal execution; and (ii) with non-blocking asynchronous I/O, Flexpath data movements may collide with application level communications, e.g., MPI communications. To alleviate these effects, we can leverage the scheduling techniques described in [13], but in the experiments shown here, the decrease in I/O overhead more than compensates for these potential side-effects of asynchronous, non-blocking I/O.

B. *Subscriptions and Metadata Distribution*

The graph shown in Figure 6 shows the time it takes for an idle subscriber to register itself with an existing data channel. In these experiments, the CNA code sits idle and waits for an application-level control message from the CSYM code. After this message is sent, CSYM idles, and CNA activates and joins the Bonds output channel. The reason we see a linear increase in registration time here is because (i) the CNA code is just a serial code that must subscribe to all data from the Bonds application, and (ii) because Flexpath uses direct connections, which requires the single CNA process to establish a connection with each Bonds publisher. The time listed here also includes the time it takes for the subscribers to send their initial data fetch requests, as outlined in Section IV, but these costs are amortized when



(a) LAMMPS (b) Lammps Helper (c) Bonds

Figure 4. Total Time spent on I/O for LAMMPS and components of analytics pipeline. The asynchronous I/O offered by Flexpath drastically reduces the time an application spends on I/O operations.

using the non-blocking calls the ADIOS read API offers.

The ability to use subscriptions and application level controls to perform such selective changes in data flow is an important feature when dealing with such large data volumes. This is because without such functionality, data would be delivered to subscribers before they need it and in addition, when they no longer need it. Considering the CSYM/CNA example, without this functionality, when LAMMPS is generating 610 MB of data, that would require nearly 1.2 GB of data to be transfered each epoch.

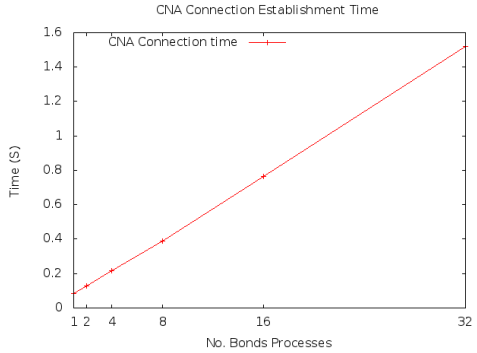


Figure 6. The time needed for 1 CNA process to join the Bonds channel.

Figure 7 shows the expected costs for collecting publisher metadata and the costs expected for the subscribers in fetching this metadata from its publisher peer coordinator. This global distribution of the metadata is one feature that allows us to employ a direct-connect model without using any external metadata services. Since these costs can potentially be induced after every epoch of data, it is important to ensure that they are kept low. At 1024 publishers and 8 subscribers, we are spending less than 15 milliseconds performing these operations. To further reduce these times, it would be possible to distribute this metadata only when changes occur.

Considering that we use subscriptions to allow subscribers to receive fine-grained slices of the published data, the overhead involved with distributing this metadata is much smaller than the overhead of possibly moving large volumes of redundant or unneeded data.

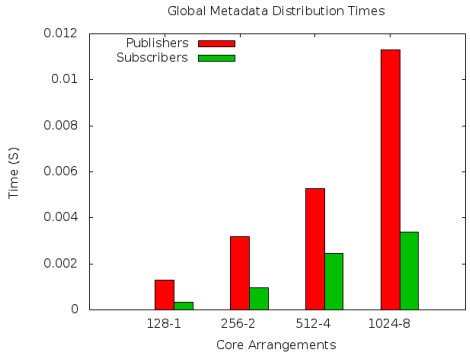


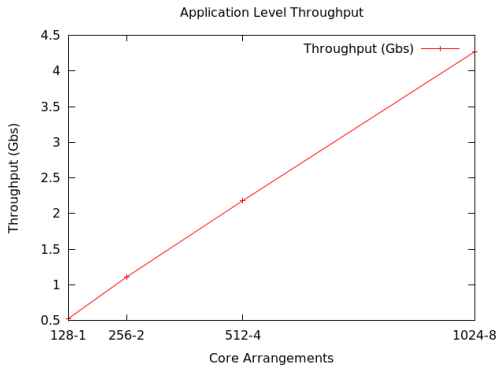
Figure 7. Time spent on collecting and distributing publisher metadata for one epoch.

C. Application Level Throughput

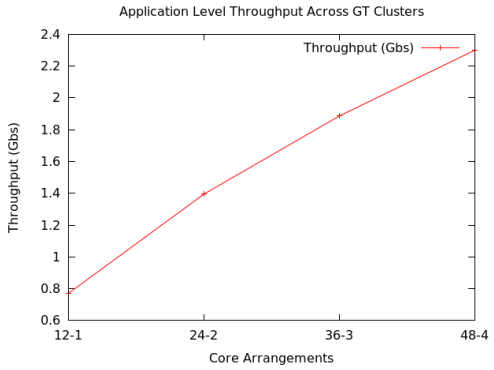
Figure 8 shows the aggregate data exchange throughput for the Flexpath system at increasing numbers of publishers and subscribers. For these experiments, we have run a two stage pipeline between LAMMPS and Lammps Helper. We conduct these experiments both on Sith and across the two Georgia Tech hosted clusters. The graphs show that in both setups, due to Flexpath's direct-connect model, we are able to achieve linear scalability as we increase the number of publishers and subscribers. This end-to-end scalability is achieved because of two key design points: (i) using subscriptions, subscribers are presented with only the slices data they request, and (ii) the use of direct connections between publishers and subscribers avoids extra data movements induced from first moving data to external brokers. Our measurements for application level throughput include the round-trip times between a subscriber's fetch request to a publisher, the data transfer time, unmarshalling costs, handler invokations, and copying the data into the user provided buffers.

VI. Related Work

Scalable pub/sub implementations created outside the HPC domain tend to consider workloads comprised of large numbers of small, potentially unrelated, messages. BlueDove [35] from IBM is an attribute-based pub/sub



(a) SITH



(b) GT

Figure 8. Application Level Throughput on Sith and across Georgia Tech clusters.

implementation for elastic Cloud-based applications, intending to use the Cassandra data store. Since it deals with small messages, it is able to benefit from the routing of messages to external dispatcher servers that also perform subscription matching before delivering the messages to the subscribers. Flexpath differs in its focus on structured, potentially complex and voluminous data events transmitted between publishers and subscribers, with its consequent use of direct connections between both. This is also the case for [30], [31], which are pub/sub systems that aim to overcome some of the inefficiencies found with routing messages and subscriptions through processing overlay networks: publisher messages are first pushed to content brokers, and subscriber subscriptions are then routed through the network to find the correct overlay node that has matching data.

In the HPC space, the work presented in [32] outlines a content-based pub/sub infrastructure layered on top of the Dataspaces [14] substrate. The work allows for introspection into the data, and subscribers can register to receive sub-samples of the events based on avg/min/max values computed from the data while it is in-flight. Flexpath differs by (1) using a direct connect model to avoid the extra data movements involved with publishing data to an external broker, as in the shared-space abstraction offered by Dataspaces; and (2) offering a subscription model that can go beyond standard array-slicing and chunking to allow publishers and subscribers to produce and consume complex data, including graphs or arrays of complex types; it also permits codes that may have type-mismatches between publishers and subscribers to exchange data.

VII. Conclusions and Future Work

This paper presents the design and implementation of Flexpath, a publish/subscribe system crafted for HPC environments. The system gives science end-users the flexibility needed to construct workflows consisting of codes that differ in their scaling characteristics, execution models, degrees of parallelism, communication patterns, and in the types and structure of data they exchange and operate on. Flexpath uses a direct connect model, has asynchronous publish and notify abstractions, and allows for fine-grained subscriptions and data transformations. It therefore, achieves good scalability and performance by reducing data movements, as data is delivered directly to the processes that request it, rather than first flowing through the external message brokers found in traditional publish/subscribe systems. Experimental results validate its scalability and the small overheads achieved with its direct-connect model. It has been released as a 'transport' in the ADIOS IO package distributed by Oak Ridge National Labs.

Future work for Flexpath is directed towards providing higher-level management constructs that allow applications and science end-users to create custom management policies for their science workflows, to better take advantage of the dynamics that occur in these complex environments, i.e., responding to dynamics along the data plane such as adaptive mesh refinement, or responding to systems level dynamics like faults, which will become more important as workflows extend beyond high-end machines and operate partially on end-user mobile devices or laptops.

References

[1] (2013) Exaos/r: Exascale operating system and runtime. [Online]. Available: https://collab.mcs.anl.gov/pages/viewpage.action?pageId=3211285

[2] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma, "In situ visualization for large-scale combustion simulations," *IEEE Computer Graphics and Applications*, vol. 30, no. 3, pp. 45–57, 2010.

[3] (2013) Visit visualization tool. [Online]. Available: https://wci.llnl.gov/codes/visit/

[4] A. Cedilnik, B. Geveci, K. Moreland, J. P. Ahrens, and J. M. Favre, "Remote large data visualization in the paraview framework," in *EGPGV*, 2006, pp. 163–170.

[5] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "Predata- preparatory data analytics on peta-scale machines."

[6] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *Euro-Par (1)*, 2011, pp. 366–379.

[7] J. S. Vetter and K. Schwan, "Techniques for high-performance computational steering," *IEEE Concurrency*, vol. 7, no. 4, pp. 63–74, 1999.

[8] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, "Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models," *Journal of Physics: Conference Series*, vol. 16, no. 1, p. 65, 2005. [Online]. Available: http://stacks.iop.org/1742-6596/16/i=1/a=009

[9] H. Childs, M. A. Duchaineau, and K.-L. Ma, "A scalable, hybrid scheme for volume rendering massive data sets," in *EGPGV*, A. Heirich, B. Raffin, and L. P. P. dos Santos, Eds. Eurographics Association, 2006, pp. 153–161.

[10] J. Dayal, J. Cao, G. Eisenhauer, K. Schwan, M. Wolf, F. Zheng, H. Abbasi, S. Klasky, N. Podhorszki, and J. y Lofstead, "I/o containers: Managing the data analytics and visualization pipelines of high end codes," in *International Workshop on High Performance Data Intensive Computing*, 2013.

[11] M. Wolf, H. Abbasi, B. Collins, D. Spain, and K. Schwan, "Service augmentation for high end interactive data services," in *CLUSTER*. IEEE, 2005, pp. 1–11.

[12] J. Lofstead, J. Dayal, K. Schwan, and R. Oldfield, "D2t: Doubly distributed transactions for high performance and distributed computing," *Cluster Computing: To Appear*, 2012.

[13] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: scalable data staging services for petascale applications," *Cluster Computing*, vol. 13, pp. 277–290, 2010, 10.1007/s10586-010-0135-6.

[14] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 25–36.

[15] M. Hereld, M. E. Papka, and V. Vishwanath, "Toward simulation-time data analysis and i/o acceleration on leadership-class systems," in *IEEE Symposium on Large-Scale Data Analysis and Visualization*, 2011.

[16] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: opportunities and challenges at exascale," in *DEBS*, 2009.

[17] H. Nasgaard, B. Gedik, M. Komor, and M. P. Mendell, "Ibm infosphere streams: event processing for a smarter planet," in *CASCON*, P. Martin, A. W. Kark, and D. A. Stewart, Eds. ACM, 2009, pp. 311–313.

[18] M. Wolf, Z. Cai, W. Huang, and K. Schwan, "Smartpointers: personalized scientific data portals in your hand," in *SC*, 2002, pp. 1–16.

[19] H. Abbasi, M. Wolf, K. Schwan, G. Eisenhauer, and A. Hilton, "Xchange: coupling parallel applications in a dynamic environment," in *CLUSTER*. IEEE Computer Society, 2004, pp. 471–480.

[20] H. Childs, B. Geveci, W. J. Schroeder, J. S. Meredith, K. Moreland, C. Sewell, T. Kuhlen, and E. W. Bethel, "Research challenges for visualization software," *IEEE Computer*, vol. 46, no. 5, pp. 34–42, 2013.

[21] J. Leibiusky, G. Eisbruch, and D. Simonassi, *Getting Started with Storm - Continuous Streaming Computation with Twitter's Cluster Technology*. O'Reilly, 2012.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2.

[23] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, metadata rich io methods for portable high performance io," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, may 2009, pp. 1 –10.

[24] S. Plimpton, R. Pollock, and M. Stevens, "Particle-mesh ewald and rrespa for parallel molecular dynamics simulations," in *PPSC*. SIAM, 1997.

[25] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam, "Gyro-Kinetic simulation of global turbulent transport properties in tokamak experiments," *Physics of Plasmas*, vol. 13, no. 9, p. 092505, 2006.

[26] P. Eugster, "Type-based publish/subscribe: Concepts and experiences," *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 1, 2007.

[27] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the kepler system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1039–1065, August 2006.

[28] J. Li, W. keng Liao, A. N. Choudhary, R. B. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netcdf: A high-performance scientific i/o interface," in *SC*. ACM, 2003, p. 39.

[29] S. K. Sahoo and G. Agrawal, "Supporting xml based high-level abstractions on hdf5 datasets: A case study in automatic data virtualization," in *LCPC*, ser. Lecture Notes in Computer Science, R. Eigenmann, Z. Li, and S. P. Midkiff, Eds., vol. 3602. Springer, 2004, pp. 299–318.

[30] A. Carzaniga, M. Rutherford, and A. Wolf, "A routing scheme for content-based networking," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 2, 2004, pp. 918–928 vol.2.

[31] G. Li, S. Hou, and H.-A. Jacobsen, "A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams," in *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, 2005, pp. 447–457.

[32] T. Jin, F. Zhang, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "A scalable messaging system for accelerating discovery from large scale scientific simulations," in *HiPC*. IEEE, 2012, pp. 1–10.

[33] G. Eisenhauer, M. Wolf, H. Abbasi, S. Klasky, and K. Schwan, "A type system for high performance communication and computation," in *e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on*, 2011, pp. 183–190.

[34] J. Lofstead, R. Oldfield, and T. Kordenbrock, "Unconventional data staging using nssi," in *In Proceedings of IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, Delft, The Netherlands, May 2013.

[35] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei, "A scalable and elastic publish/subscribe service," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1254–1265, 2011.