Scibox: Online Sharing of Scientific Data via the Cloud

Jian Huang†, Xuechen Zhang†, Greg Eisenhauer†,
Karsten Schwan†, Matthew Wolf†*, Stephane Ethier§, Scott Klasky*
† *Georgia Institute of Technology,* § *Princeton Plasma Physics Laboratory,* * *Oak Ridge National Laboratory*
{*jhuang95, xczhang, eisen, schwan, mwolf*}*@cc.gatech.edu,* {*ethier*}*@pppl.gov,* {*klasky*}*@ornl.gov*

***Abstract*—Collaborative science demands global sharing of scientific data. But it cannot leverage universally accessible cloud-based infrastructures like DropBox, as those offer limited interfaces and inadequate levels of access bandwidth. We present the Scibox cloud facility for online sharing scientific data. It uses standard cloud storage solutions, but offers a usage model in which high end codes can write/read data to/from the cloud via the APIs they already use for their I/O actions. With Scibox, data upload/download volumes are controlled via D(ata)R(eduction)-functions stated by end users and applied at the data source, before data is moved, with further gains in efficiency obtained by combining DR-functions to move exactly what is needed by current data consumers. We evaluate Scibox with science applications and their representative data analytics – the GTS fusion and the combustion image processing – demonstrating the potential for ubiquitous data access with substantial reductions in network traffic.**

***Keywords*-Cloud Storage, Data Sharing, Scientific Data**

I. Introduction

Global, distributed scientific processes critically rely on the data generated by scientific simulations and instruments. Examples range from investigations by science teams in domains like fusion modeling [1] or in combustion research [2], [3], to widely distributed sets of researchers and amateurs in astronomy [4], to a plethora of enterprise applications able to benefit from widely shared data like SmartGrid or SmartCity projects. Common to all such endeavors is the need for convenient and ubiquitous data sharing, which the scientific community has pursued by constructing extensive grid-based data sharing infrastructures [5] supported by high end networks within and across national and international science facilities research labs [6]. Concurrent with these developments, businesses have created their own infrastructures for conveniently sharing data across large numbers of widely distributed participants, like the DropBox [7], GoogleDrive [8], and iCloud [9] services used across the globe.

There are cost and overhead issues with directly using commercial data sharing facilities like DropBox for scientific data exchange. First, for the tens of Terabytes of data generated by petascale science simulations (e.g., GTS [10], [11], LAMMPS [12]) per day [10], even if it were possible to move all of that data to the cloud, storage cost would quickly exceed science budgets, in lieu of the storage pricing of standard storage provided by Amazon S3. This means that storage costs would be $972.8 per day, at minimum, where data transfer costs are linear with the amount of data moved through the Internet. Second, scientific data is usually stored in a storage hierarchy which includes both memory of I/O staging nodes and disks of storage servers. Moving data from disk to cloud storage can incur a very long latency when data size is in the scale of Gigabytes, let alone Terabytes. User experience could be worse if the data does not contain interesting contents as networking bandwidth and CPU cycles are wasted dramatically. Third, the existing interfaces provided by cloud storage service like Dropbox for data sharing limits itself in its ability to provide content-aware raw data or partition of data which users have real interests so as to reduce the cost of using cloud storage. Users cannot effectively express their constrains on data contents.

This paper explores cloud-based data sharing and storage challenges and opportunities (i) for simulation output data processed on the high end analytics or visualization engines in petascale science facilities, and (ii) for the outputs generated by large scientific instruments. We adopt the ideas underlying existing cloud data sharing facilities like DropBox, in terms of their ease of use and universal accessibility, but address key issues critical to making them usable for scientific data exchange. First, for constrained data exchange, for the tens to hundreds of gigabyte data volumes generated by high end simulations, we permit data sharing to focus on the data of interest to end users, where such 'interest' depends on data contents expressed by user-defined analysis methods, i.e., instead of indiscriminately sharing raw data, data sharing facilities must offer methods for constraining cloud-based data exchange and storage. Commercial facilities like DropBox do not yet offer such functionality. Second, for science data and its metadata-rich storage formats like HDF5 [13], BP [14], etc., we exploit such metadata for opportunities to filter and reduce data as per end user interests and needs. Third, we encourage online methods to constrain data exchanges, to permit end users to focus on the data important to the current scientific investigations being pursued.

The Scibox infrastructure described in this paper implements methods for the online sharing of scientific data across shared cloud resources. It leverages the ease of use and universal accessibility of commercially developed cloud data sharing software backed by large-scale cloud stores, like Amazon's S3 [15], but enhances open source cloud-based data sharing services with new functionality for better access to and use of the large volumes of structured scientific data employed in scientific inquiries. Specifically, Scibox extends the limited interfaces of systems like DropBox to better serve science users, and it provides novel methods to cope with the inadequate levels of ingress and egress bandwidths available to/from the remote cloud stores in which data is maintained. Scibox (1) proposes a science data usage model and (2) offers methods to circumvent unnecessarily large data uploads to or downloads from the cloud. Concerning (1), Scibox presents to data producers and consumers the standard I/O APIs already

used by science applications, like the Adaptive I/O system (ADIOS). As a result, science codes can write output to the cloud that can then be directly read by subsequent, potentially remote data analytics or visualization codes, in the same fashion as I/O and subsequent analysis are being performed in today's high end facilities used for running science simulations (e.g., at ORNL, LLNL, etc.). Concerning (2), to reduce cloud data upload/download volumes, Scibox permits an end user to identify the exact data needed for each specific inquiry (i.e., analytics activity), by specifying the D(ata)R(eduction)-function that is applied at the data source and before data is actually uploaded to the cloud. In addition and for efficient online data sharing across multiple concurrent science end users, Scibox then combines users' different DR-functions into a cumulative data reduction method, in order to upload to the cloud only those data items needed by the complete current set of data sharing clients.

Scibox realizes its data sharing approach by leveraging the metadata-rich descriptions of scientific data: when data is first generated by a data provider, only its metadata [13] is placed into the cloud. Data consumers specify DR-functions against such metadata, to identify the data subsets and transformed data items they desire. These consumer-driven inquiries, then, give rise to actual data movements into and out of the cloud, thereby limiting data transmissions only to those items actually needed by data consumers. The additional step performing function merging across multiple concurrent clients aims for minimal in-cloud data sizes for current sharing patterns.

Beyond Scibox's novel science data-centric functionality, additional advantages of using it vs. commercial methods like DropBox include the following. First, when using Scibox backed up by say, Amazon's S3 store, science users can take advantage of that store's high durability, availability, and its pay-as-you-go pricing model [15], [16], rather than relying only on increasingly cost-stressed internal research facilities. Second, by carefully managing the actual data stored in the cloud, via DR-functions, scientists can limit their exposure to storage costs imposed by providers, by storing only those items actually needed by their specific inquiries.

We envision a use of Scibox in several concrete contexts. For wide area data sharing via commercial storage like Amazon's S3, Scibox essentially acts as an intermediary that helps orchestrate data flows from some source, like a supercomputer center's analytics machines with access to its large-scale storage, to some number of sinks, such as scientists inspecting data from their last simulation run. The science applications running on high end machines and producing source data use standard methods to store their data locally and in ways accessible to Scibox, with additional data sharing actions possible as long as data remains present at the source. A second use case is data generated in high end experimental facilities, where local storage at those facilities contains the large data volumes being produced and Scibox is used to share select data items with remote collaborators. In both cases, however, once data has been shared and is therefore, present in the cloud, it can be inspected and viewed any number of times and will remain present in the cloud until explicitly deleted by end users.

The use cases and benchmarks employed to evaluate Scibox mirror the way in which we expect this functionality to be used. Micro-benchmarks evaluate basic overheads experienced at source and sinks. The applications run are (1) scientific simulations producing outputs suitable for online analytics and visualization, and (2) experimental data produced by an instrument, and then consumed by various analytics clients. For both cases, the numbers of clients used will scale for larger science collaborations. We conduct extensive evaluation using both synthetic and real scientific workloads, including GTS physics diagnosis and combustion image processing. Our results show that Scibox can reduce networking traffic by up to 65x. In addition, Scibox users can have a much better cloud experience with less I/O latency and significantly saved usage cost.

In summary, Scibox makes the following contributions:

Detailed Meta-data. We store both metadata about files and a metadata list about file contents, e.g., variable names, in the cloud, and such information is pushed to users who register for the Scibox service. It is such metadata availability that makes it possible to identify and then limit the actual data moved into/out of the cloud to service end users' specific analytics needs.

Data Filtering using DR-functions. DR-functions permit users to specify data subsets of specific interest to them. DR-function can be as simple as stating a range of offsets in some array dimensions or as complex programs for data transformation and compression. DR-function are registered and realized at runtime, whenever needed for analytics tasks undertaken by end users.

Efficient Data Sharing. Only the data that satisfies the registered criterions, expressed via DR-functions, are uploaded into cloud storage for sharing with data consumers. Overlapping data sets for different consumers are merged to further reduce data redundancy on the storage server and data volumes transferred across the network.

The remainder of this paper is organized as follows. Section II discusses related works. Section III describes the system architecture of Scibox. Section IV carries the design and implementation of DR-functions. Section V discusses the key ideas for efficient data access and metadata management in Scibox. Section VI presents other implementation issues of Scibox. Section VII presents our experimental results, and Section VIII concludes the paper.

II. Related Works

A. Enterprise Data Sharing via the Cloud

Enterprises routinely use cloud-based data sharing solutions like Google Drive [8] and Dropbox [7]. For both, since the cost of additional cloud storage increases linearly with the amount of data [17] and the number of users [18], neither system offers a cost-effective solution for the large data sharing needs of high end science instruments or simulations. Scibox, therefore, is designed to use commercial cloud stores, but

it can also operate with storage owned or provided by end users and certain science communities. In addition and for both, it exploits the structured nature of science data sets by using a metadata-rich storage format suitable for use with D(ata)R(eduction)-functions.

B. Infrastructures for Scientific Data Sharing

Rapid transfers of science data via networks has been explored in much previous work, including by GridFTP [19], Globus online [20], and others [21]. Notable features for improved performance and reliability include parallel and striped data transfers, automated restarts, and probabilistic methods for selecting the network links across which data is sent [21]. All such efforts [20], however, assume data to be stored in local storage systems, and are therefore, unable to benefit from commercial cloud stores' data durability, reliability, and scalability. This has been recognized by recent efforts at SDSC (San Diego Supercomputer Center) [16] using OpenStack's Swift Object Store to provide cloud storage for academic and research partners, following the pay-as-you-go pricing model. The effort demonstrates the feasibility of using commercial cloud storage technology for science data. To that work, Scibox adds functionality that includes its use of the metadata-rich IO APIs and the subsequent use of that metadata to reduce data up/download volumes.

C. Speeding up Network Data Transfers

As mentioned above, GridFTP [19] seeks to use parallelism to improve data transfer rates, but it also provides an I/O interface with which users can ask for subsets of data files at a specified offset. Scibox expands that functionality by providing users with a metadata-rich API that provides information about variable and array names, rather than burdening them with the need to know about low-level concepts like offsets in files. This said, there remains substantial room for improvement of the data transfer speeds. Finally, repeated accesses to the same data via Scibox can also benefit from cached Internet content via CDN technology [22]. Additional improvements in the functionality of our software may be derived from using content-based data indexing like those used by FastQuery [23] or in SciDB [24], but we have not yet investigated the potential benefits of using such approaches – in terms of additional metadata up/downloads and storage vs. the consequent, potential gains in source-based data selectivity.

III. Scibox System Overview

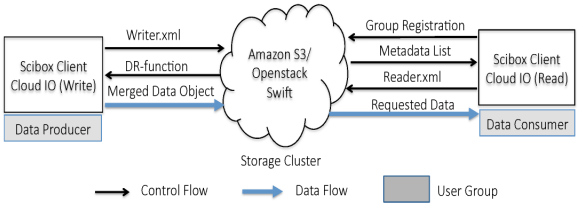

Fig. 1. Scibox architecture

Key to Scibox is the simple question of: *how to design a scientific data sharing infrastructure so as to reduce data transfer and storage sizes?*, and the approach taken to answer this question is one that *stores only the data desired by end users in the cloud.* The Scibox realization of this solution approach is shown in Figure 1, along with its control and data flows. The figure shows clients able to provide DR-functions used at sources to upload only the data clients desire. It also indicates function merging for Scibox's built-in DR-functions when different clients request overlapping data.

The current implementation of Scibox uses the object store to implement data storage on some **storage cluster**. This means that the generated scientific data are stored as objects that can be accessed by users through unique URLs. The **data producers** are scientific simulations or instruments, which repeatedly and/or periodically produce output data to which users apply various analytics actions/codes. We assume such data to be locally stored and available for some time, and we assume the presence of producer-side compute capacity for data cleaning, preconditioning, and for the analytics needed to select/filter data, thus preparing it for the broader degree of data sharing enabled by Scibox. **Data consumers** download desired data from the cloud, where a **user group** is composed of some single data producer and multiple consumers who share select content. Multiple groups can be created for sharing different data sets produced by an application or instrument. For example, when using the GTS fusion modeling simulation as a data producer, the simulation dumps checkpoint, diagnosis, and visualization data into different files, naturally mapped to different user groups interested in such content. For all data sharing, however, Scibox client functions are running on both the producer and consumer sides, the latter enabling the data reductions desired by end users. The Scibox control flows, therefore, span producers and consumers.

```
<?xml version="1.0"?>
<adios-config host-language="Fortran">

  <adios-group name="restart" coordination-communicator="comm">
    <var name="mype" type="integer"/>
    <var name="numberpe" type="integer"/>
    <var name="istep" type="integer"/>
    <var name="MIMAX_VAR" type="integer"/>
    <var name="NX" type="integer"/>
    <var name="NY" type="integer"/>
    <var name="zion0_1Darray" gwrite="zion0" type="double"
         dimensions="MIMAX_VAR"/>
    <var name="phi_2Darray"   gwrite="phi" type="double" dimensions="NX, NY"/>
        <!– for reader.xml –>
    <rd type=8 name="phi_2Darray" cod="int i;\n\double sum = 0.0;\n\for(i = 0;
             i<input.count; i= i+1) {\n\sum = sum + input.vals[i];\n\}\n\return sum;\n\"
```

Fig. 2. An example of the writer.xml and reader.xml files for the GTS application, composed of zion and phi data array variables.

Scibox Producers The functionality of Scibox is run by daemon processes located on the analytics machines/clusters present in leadership facilities (e.g., on the Smoky cluster at ORNL). This means that HEC machines running simulations continue to write output data to the high end parallel file systems attached to those machines, the Scibox daemon running on the analytics machine reads desired data from the parallel file system, thereby enabling it to subsequently share select data via the cloud.

Operationally, creating a user group requires its producer to store a *writer.xml* file in cloud, as shown in Figure 2 for the GTS application. The XML file, which is composed of variable names and its related attributes, is sent to every

Type	Description	Example
$DR1$, $DR2$, $DR3$	Max(variable), Min(variable), Mean(variable)	Max(var_2Darray), Min(var_2Darray), Mean(var_2Darray)
$DR4$	Range(variable, dimensions, start_pos, end_pos)	Range(var_int_1Darray, 1, 100, 1000)
$DR5$	Select(variable, threshold1, threshold2)	select var.value where var.value $\in$ (threshold1, threshold2)
$DR6$	Select(variable, DR_Function1, DR_Function2)	select var.value where var.value $\geq$ Mean(var)
$DR7$	Select(variable1, variable2, threshold1, threshold2)	select var2.value where var1.value $\in$ (threshold1, threshold2)
$DR8$	Self defined function	double proc(cod_exec_context ec, input_type *input, int k, int m) { int i;int j; double sum = 0.0; double average = 0.0;for(i = 0; i;m; i= i+1) sum = sum + input.tmpbuf[i+k*m]; average = sum / m;return average; }

TABLE I
DESCRIPTION OF *DR*-FUNCTIONS

consumer in the form of a metadata list, thus enabling them to determine and specify the customized data subset selections they desire. As stated earlier, such specifications are via DR-functions determined by clients, with additional detail about these functions presented in Section IV. After processing the DR-functions on the original data, the outputs having overlapping data sets are merged and written to cloud storage using the cloud I/O transport and object storage interfaces.

Scibox Consumers Each data consumer is assigned a unique *User ID*, and it is registered with some specific user group. After reviewing the metadata list regarding the group, a consumer creates a XML file *reader.xml* similar to *writer.xml*, but with variable names that specify the desired data subsets and stating a DR-function attribute. If the DR-function attribute is not specified, the full datasets will be pushed to the user, else those functions are used for producer-side data filtering and/or transformation. As with producers, Scibox clients for data consumers are executed as daemons, with the daemon periodically checking the consumer's XML file for changes in datasets and DR-functions, and checking cloud storage for data updates, the latter leading to downloads of the latest version of desired data via the cloud I/O transport.

IV. DR-FUNCTIONS

A DR-function transforms data as per end user instructions, with useful functions including those that reduce data and prepare it for cloud storage and remote access. DR-functions can be explicitly programmed by end users, generated from higher level descriptions, or created automatically by deriving them from users' I/O access patterns, such as repeated accesses to certain variables.

A technical issue with using DR-functions is their requirement for producer-side computational capacity. While such capacity is likely available in the leadership facilities in which high end simulations are performed, it may not be present with certain instruments, an example being the combustion instrument used in this paper's experiments. This means that when a user group has a large number of data consumers, like an entire school class, the execution of individual DR-functions, one-by-one, can impose unacceptably high computational overheads on producers. To address this issue, Scibox offers function combining methods for its basic DR-functions and their derivatives in the DR-function library. For such functions, if multiple consumers require the same DR-function, the function will only be executed once and its output data will be reused for multiple consumers. The current Scibox prototype supports DR-functions classified into eight categories, described in detail in Table III. Functions of types $DR1$ to $DR4$ – *basic DR-functions* – describe user requirements regarding a single variable. Functions of types $DR5$ to $DR7$ describe more complex relationships between variables. For example, a user of GTS data can take advantage of $DR7$ when the ion's temperature data is needed only when its velocity is larger than some threshold. Finally, more advanced users can explicitly define their own, custom DR–functions – Type $DR8$ – using the Co(n)D(emand) [25] programming language. CoD is used because its simple code generation facilities can be run at consumer, producers, or 'in' the cloud, across the entire set of participants in a Scibox system. The system operates by registering a string describing the DR-function at data producers, then compiling and running the function 'on demand' at the producer, on the specified input data. For such custom functions, Scibox does not guarantee them to be executable for arbitrary input data, e.g., if there are mismatches in the function's assumptions concerning input data types and sizes with the actual data seen, function execution will fail, returning the original data to the user.

The implementation of DR-functions $DR1$ to $DR7$ leverages the *reader.xml* file, to which any data consumer can add XML attributes about DR-function types and their input parameters. The system parses this file, generates DR-functions, and then applies them to the input buffer, for all functions specified. As stated earlier, however, since Scibox may need to support hundreds of data consumers with individually customized data requirements, executing their DR-functions one-by-one can be costly. In response, Scibox merges all basic DR-functions, e.g., max, min, average, and subsets of data arrays, and then runs the composite function as a single scan across its input buffer. The same technique can be used for complex DR-functions if they can be decomposed into sets of basic functions. For example, $DR6$ is implemented based on $DR3$, the $DR3$ functions can be stripped out from $DR6$ to merge with other basic DR-functions. An optimization to improve DR-functions in minimizing their potential effects on producer performance is to define them as best-effort, which means that they can be disabled at any time. Data consumers will only experience consequent delays in data updates due to perhaps, increased network bandwidth requirements. Finally, for DR-functions of type $DR8$ implemented using CoD [26], [25], [27], when $DR8$ is specified in the XML file, on the producer's side a string representing the body of the DR-function and the address of data buffer for processing are passed to CoD context. The string is then dynamically compiled into a binary executable. Table III shows an example

of $DR8$ for calculating the average of row values of a matrix.

V. Storage Objects and its Management in Cloud

Scibox uses OpenStack Swift as its object based storage system. Objects are stored in three different containers: (1) the *Group Metadata Container* (GMC), (2) the *User Metadata Container* (UMC), and (3) the *Data Container* (DC). The GMC stores the XML files of producers and consumers as objects. Producers periodically download the consumers' XML files for DR-functions from GMC, and consumers read producers' XML files on group registration for obtaining the metadata list regarding a specific file. The UMC stores the objects having metadata information regarding each user's requested data, including time step, URL to a data object, and data range in the object if partial object access is supported. All of the data objects are stored in the DC. With this design, a data sharing intent is carried out by simply adding a URL to a user's metadata object in the UMC. To quickly convert the consumer's file name in XML file to a URL of an object in cloud, we use the multi-level index approach to manage both metadata and data objects for Scibox. For every user group, it maintains a *Super File* which is responsible for indexing the GMC and the user's metadata in the UMC, as shown in Figure 3. The Scibox client daemon for consumers can download expected data through the URL of a data object and then convert it into desired local file formats, e.g., BP, for its own local storage.

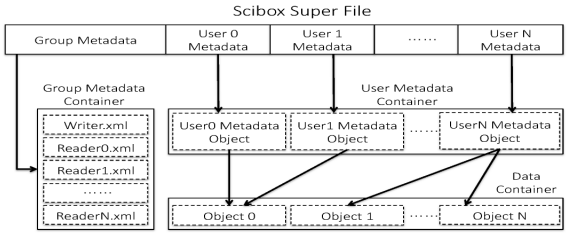

Fig. 3. Scibox data management in cloud storage

A. Determination of Object Size

With the design for object sharing in Scibox, all overlapped data sets generated by DR-functions must be merged before writing to cloud storage, and every stored object can be used to serve different users. However, users have to download the whole object, even if only a small portion of its data is needed, since the current Amazon S3/Swift stores do not support partial object access. This can significantly increase a user's costs for downloading from the cloud, particularly for larger object sizes. This issue could be addressed by simply using small object sizes, but such a choice would in turn increase object management overheads due to large numbers of objects and lower network performance due to small message sizes.

To solve this issue for private cloud users, we have instrumented the Swift software by adding a partial object access protocol, which is presented in Section VI. With this issue being solved, the object size can be determined based on network performance T, which we found in experiments, to vary widely with different object sizes $Size_{net}$ and numbers of producer threads. We find fitting functions $T = f(Size_{net})$ under different thread concurrency with offline profiled data. And the function runs online to predict the optimal object size.

We note that a partial object access protocol is not possible to use with public clouds like Amazon's S3, as we cannot change its software stack. For using public clouds, we must therefore, determine an object size $Size$ that reduces the cost incurred from downloading unneeded data to be less than the cost savings arising from object sharing. More formally, assume that the storage pricing of standard cloud storage is $\$\alpha$ per GB, the pricing of data transfer into cloud is $\$\beta$ per GB, and the pricing of data transfer out from the cloud is $\$\gamma$ per GB. For example, α, β, and γ are 0.095, 0, and 0.012, respectively, for Amazon S3. The requested data sizes for n consumer clients are $Data_0, Data_1, ..., Data_n$. Without data sharing, the associated costs of using the cloud $Cost_{nosharing}$ can be defined as:

$$Cost_{nosharing} = (\alpha + \beta + \gamma) * \sum_{i=1}^{n} Data_i; \quad (1)$$

Assume that the n data sets can be merged into a single data object, of size $Size$, then the associated costs for sharing the objects with consumers can be described as $Cost_{sharing}$:

$$Cost_{sharing} = \alpha * Size + \beta * Size + \gamma * \sum_{i=1}^{n} Size; \quad (2)$$

Since $Cost_{sharing}$ should be less than $Cost_{nosharing}$, then we can have

$$Size \leq \frac{(\alpha + \beta + \gamma) * \sum_{i=1}^{n} Data_i}{\alpha + \beta + n\gamma}. \quad (3)$$

This inequality implies that using any object sizes that are smaller than $Size$, Scibox will lead to benefits obtained from object sharing. Therefore, for cloud storage with partial object access supported, the object size depends on $Size_{net}$, otherwise, we determine that if $Size$ is larger than $Size_{net}$, then object size is set to $Size_{net}$, whereas if $Size$ is smaller than $Size_{net}$, then object size has to be set to $Size$, the latter to make tradeoff between cloud cost and network performance.

B. Merging Requested Data

Given the object size and cost model discussed in Section V-A, Scibox needs to merge the output of DR-functions that have overlapping data subsets. We classify DR-functions into three groups. 1) The DR-functions with functional overlap, i.e., $DR1$, $DR2$, and $DR3$ functions, need only be executed once if they operate on the same data set. 2) The DR-functions resulting in data subset overlap, like the $DR4$ functions, can be merged according to the values of the start position of each requested data range pair. 3) The DR-functions in the independent group cannot be merged, so they are executed concurrently on the data producer side, subject to restrictions to function dependencies, e.g., for some iterative DR-functions, in which the output of one function is the input of another function, we have to sequentially schedule their executions to guarantee the correctness of the requested data.

VI. Implementation Issues

We implemented the prototype of Scibox with the OpenStack Swift object store [28] software and the ADIOS [14] library. This section describes how the ADIOS library interacts with Swift storage cluster, and it discusses improvements to Swift for partial object access. It is noted that the scientific data usage model proposed in Scibox can be easily applied in other I/O libraries.

A. Cloud-IO: Yet Another Transport Method in ADIOS

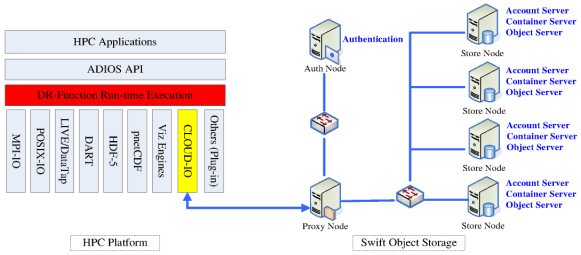

Fig. 4. Software organization of CLOUD-IO transport

The ADIOS library supports multiple I/O transports, e.g., MPI and POSIX, for users with different I/O requirements. Such transport methods are configured in an XML file, so that a change in transport does not require developers to modify or recompile their applications. Scibox contributes to ADIOS a new Cloud-IO transport so that I/O requests can be sent to the cloud with the same APIs already used by end users for constructing their I/O pipelines on high end machines. The transport is implemented so that application-level data buffers can be directly used as inputs for Scibox's DR-functions.

The software architecture presented in Figure 4 depicts a cloud I/O library that can perform all object related operations, such as user account authorization and verification, container/object creation and destruction, and object uploading and downloading. The Cloud-IO transport simply calls its functions, e.g., *object_upload()* and *object_download()*, to handle cloud I/O requests. The cloud-IO transport operates both with POSIX IO and MPI programs for convenience and/or to exploit parallelism for obtaining improved cloud upload bandwidth [29].

B. Partial Object Access in Swift

To reduce data redundancy in cloud storage, we have instrumented Openstack's Swift to support partial object access. Consider, for instance, N users sharing the same data object, of size is S, where each user only requires p_i $(0 \leq p_i \leq 1)$ percent of the data. As discussed in Section V-A, without partial object access, an additional $S * \sum_{i=1}^{N}(1-p_i)$ amount of data traffic will be incurred, which is inversely proportional to p. Partial object access can avoid those additional cost, while also providing a flexible interface for data access in cloud storage. Unfortunately, current commercial cloud storage platforms make it difficult to deploy such functionality. An alternative approach we have not yet evaluated is to also pay for running a virtual machine in EC2, then run data reduction and merging functions there, at the additional cost of paying for VM cycles.

The Scibox implementation of partial object access alters the open-source Swift object store. As shown in Figure 4, Swift is comprised of proxy node, authorization node, and storage nodes. The proxy node accepts all incoming requests and forwards them to corresponding storage servers. Before communicating with storage nodes, users are required to authenticate against the authentication service running on the authorization node for connection parameters and token. The account server, container server, and object server are configured to run on the storage nodes. The object servers will process all object operations including put, get, and delete. To support partial object access without influencing Swift's normal operations, we add two parameters to the call interfaces to indicate the data range of the partial access, and we create a new function *ParGET()* on the object server side. Bounds checking and checksum methods are used to verify the correctness of the retrieved data.

VII. Performance Evaluation and Analysis

The performance of Scibox is evaluated in a private cloud comprised of multiple clusters located across multiple campuses, as shown in Figure 5. Jedi runs the Swift store, acting as the cloud service provider, with three nodes and a total disk capacity of up to 2.4 TB. The Vogue cluster, which has 10 eight core servers, is used to run applications that produce scientific data using the ADIOS library. It plays the role of an analytics cluster in supercomputing facility, able to access simulation produced by high end machines but also connected to the cloud. An alternative data producer is 'Aero', which stores and processes 10,000 images produced by an experimental combustion facility, which can be shared with students for their specific research purposes. To simulate data access with different network latencies via the Internet, we run two remote Scibox consumer clients at Wayne State University, Detroit, Michigan, and at The Ohio State University, Columbus, Ohio, together with local clients run at Georgia Tech, in Atlanta, Georgia. We use the latest stable version of OpenStack Swift and for cost analysis, we use a pricing model similar to Amazon's S3 [15].

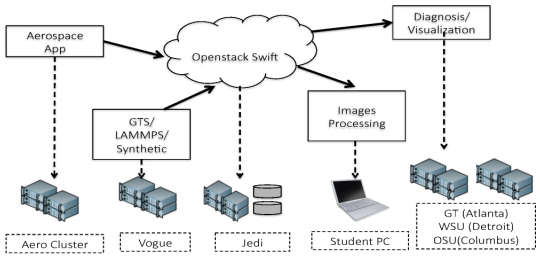

Fig. 5. Experimental setup with multiple clusters and scientific applications and their analytics.

A. Synthetic Workloads

This section uses synthetic workloads to evaluate basic characteristics of Scibox. In particular, we model ten variables shared by multiple consumers in 1000 requests, where requests are uniformly distributed in eight types corresponding to those in Table III. Note that this is likely somewhat pessimistic in

terms of typical sharing behavior, since for realistic large-scale simulation runs, there is often data of particular interest to science users and thus, more widely shared. Examples of popular data can include flame fronts in combustion simulations, turbulence in fusion simulations, data pertaining to crack formation in materials modeling, etc. Our future work will experiment with zipfian distributions to model such sharing, but for our current synthetic sharing scenario, we use a workload generator that simply randomly selects proper input values, e.g., *start_pos* and *end_pos* for $DR4$, *thresholds* for $DR5$, and the variable to access for a request given the size of the variable. We also implement three different type-8 DR-functions, including FFT, histogram diagnosis, and average of row values of a matrix, for the actual science workloads used in experiments. Most experiments use one data producer serving the 1000 requests from four clients servers.

1) Performance with Different Variable Sizes: Experiments first vary the sizes of the ten variables, to explore the degree of data reduction obtained for both cloud upload and download, compared with directly accessing the raw data via the cloud (e.g., if science users simply use DropBox). As shown in Figure 6, Scibox achieves an overall traffic reduction of up to 70x, with an average of 65x. This is because (1) for most DR-functions, the output data size returned to consumers is at least an order of magnitude smaller than the raw data volume, (2) although Type-4 DR-functions still have to upload a significant amount of data to cloud storage, with the merging algorithm and with object sharing, Scibox only maintains one copy of the accessed data, and the size of that data is also much smaller than raw data otherwise sent to each client (the reduction factor of Type-4 DR-function is 43x on average).

We present the request service times and performance improvements of raw data sharing vs. Scibox in Figure 7. Overall performance speedup is 8.5x, which is smaller than performance speedup seen for $DR4$ requests. This is due to differences across DR-functions. For instance, for 64 MB size variables, Type-4 and Type-8 DR-functions achieve 33x and 1.02x speedups, respectively, but since DR4 requests only account for a small percentage of total requests, overall speedup is compromised.

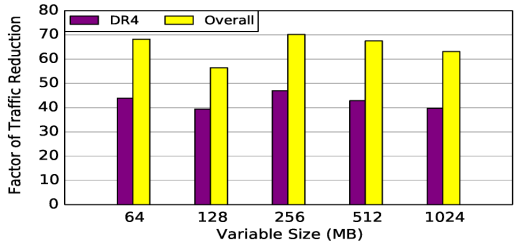

Fig. 6. Factor of traffic reduction with Scibox compared to a vanilla cloud storage system.

2) Performance Impact of Object Size: After merging overlapping data subsets, Scibox must upload the resulting data to the cloud, taking into account both network performance and the cost of cloud storage. This section studies how object sizes impact performance and the cost of using Scibox. For experiments, eight producers are executed on eight cluster

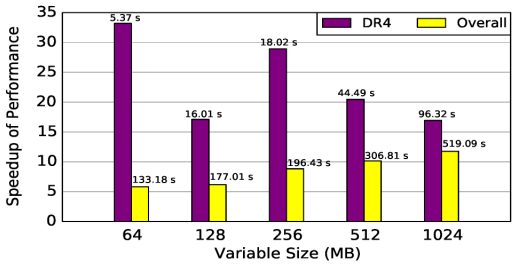

Fig. 7. Speedup of performance and request service times with Scibox over a vanilla cloud storage system.

nodes, each processing 1000 requests. The Scibox producer daemon partitions the merged data according to different object sizes for cloud uploading. As shown in Figure 8, the object size is increased from 2 MB to 256 MB. Uploading with a larger object size can reduce network transfer overheads, resulting in reduced request execution times. But with cloud storage not providing partial object access, the system may lose cost-effectiveness as more redundant data is stored and then downloaded by consumers. As Scibox uses the cost-performance model when calculating the object size, from the same figure, we can observe that it can automatically and dynamically select an appropriate object size, considering the request's characteristics, thus achieving a balance between cost and performance.

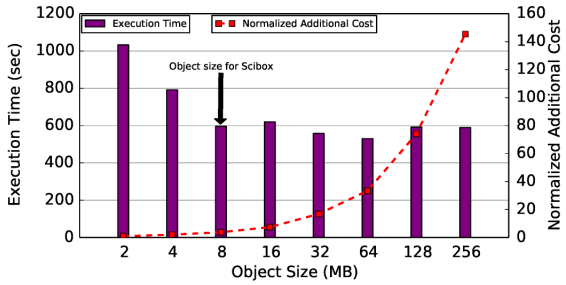

Fig. 8. Execution time with different object size

3) Scibox System Scalability: This section evaluates the scalability of Scibox for increasing numbers of producers and consumers. We evaluate the producer's PUT and consumer's GET performance separately, since the two operations are handled asynchronously in Scibox.

On the producer side, we increase the number of producers from 1 to 16, each of which needs to transfer 125 GB data to the cloud. We compare the total execution time of the PUT operation from the time when data is generated to when data is ready to be pushed to users, in the stock system without merging vs. in Scibox with merging. From the results shown in Figure 9, it is apparent that both systems produce increased execution times as system loads are increased at the producer side. The stock system does not scale well, however, with a super-linear increase, because of limited resource availability on storage servers. With Scibox, since data is merged before uploading, only 3.62 GB total data is uploaded to the cloud, at 16 producers, compared with 204.05 GB in the stock system. This results in performance speedup for Scibox of 9.7x on average over that of the stock system.

We next increase the number of requests processed by every producer, from 1000 to 8000, with four producers. The total

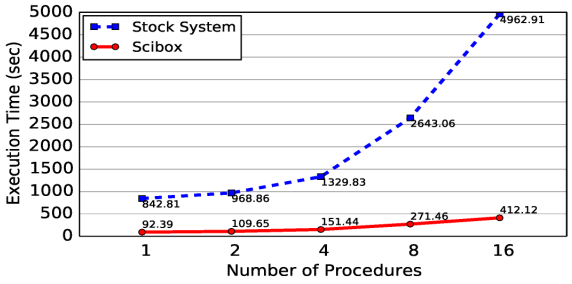

Fig. 9. Performance of PUT with increasing numbers of producers.

execution times of the PUT operation in the stock system and in Scibox are shown in Figure 10, respectively. Our observation is that with smaller numbers of producers, the performance of the stock system has a close-to-linear behavior as the number of requests is increased. With Scibox, execution time can be sublinear with the number of requests, indicating its superior scalability. The performance speedup of Scibox is 10.95x on average over the stock system.

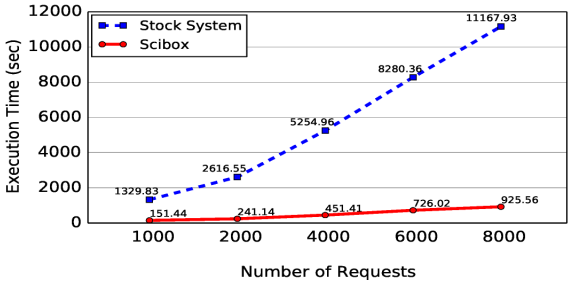

Fig. 10. Total execution time of PUT as the number of requests per producer is increased from 1000 to 8000.

On the consumer side, we measure user-experienced latency to study the scalability of Scibox. Specifically, we set up a user group with four producers and an increasingly larger number of consumers, from 2 to 64. Each consumer generates 1000 requests. We measure the user-experienced end-to-end latency from submitting their requests until request completion. Figure 11 shows the results of the stock system vs. Scibox. Latency is reduced by up to 3.73x and 2.99x on the average, and the latency curve with Scibox has a smaller slope than that of the stock system, again demonstrating improved scalability for increased numbers of consumers sharing the same data. This is because Scibox supports partial object access, which effectively reduces the data set size in memory and improves caching efficiency. In addition, the overhead for management of the raw data is also significantly reduced as its size is reduced by up to 64x.

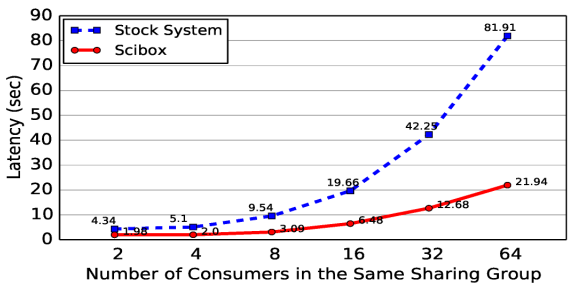

Fig. 11. Performance improvement of partial read

B. GTS Workloads

The GTS petascale application [11] is a particle-in-cell code designed for studying turbulent transport in magnetic fusion plasmas. Every few time steps, multiple variables are written to files for physics diagnostics, visualization, or checkpointing. In this experiment, we focus on sharing data in the checkpointing file restart.bp. The file includes all the variables necessary to restart a simulation, the most important of which are the $zion$ and phi arrays. $zion$ contains the pase space coordinates of all the simulated ion particles while phi contains the grid-based electrostatic potential field. All the kinetic information is held by the charged particles while the turbulence and all the waves created by the collective interactions between the particles are contained in the electrostatic field. Analytics are performed by Scibox consumers to calculate the particles' spatial distribution based on the $zion$ arrays, and to execute Fourier transforms or histograms for physics diagnosis on phi. By default, the two arrays are stored together along with many other variables in the file restart.bp. With Scibox, the consumers can filter out the data which are not needed by the analytics using DR-functions, e.g., **I (zion, *DR*1, MAX)** and **II (phi, *DR*8, Histogram)**.

We run the GTS application on the Vogue cluster with 128 parallel processes and 5 particles per grid cell. The total amount of raw data that are generated by GTS for software restart is 908 MB for the chosen problem size. We set up 30 Scibox consumers to download the data from three locations. Every consumer requested both $zion$ and phi variables using either DR-function I or II. Ten send requests from Wayne State University (WSU); another ten are located at The Ohio State University (OSU); and the remaining consumers are placed locally in Atlanta (GT). Because of the firewall issue, data cannot be transferred from Jedi to their remote clients using curl, instead we use FTP for data transfer. For Scibox the related networking latency is calculated based on the reduced data size and its measured networking bandwidths, which are 900 KB/s, 4.4 MB/s and 44 MB/s from the Jedi object store to WSU, OSU, and GT, respectively. In experiments, we assume that the data are directly pushed to a user's local storage, as soon as it is generated. Therefore, user experienced latency includes the GTS computation time, the DR-function execution time for Scibox users, data downloading time, and post-computation time for non-Scibox users. Results are shown in Figure 12.

Several interesting observations may be made. First, Scibox consistently achieves better performance than FTP-based, direct transfer solutions, because only the results of DR-functions are transferred via the cloud. For $zion$ and phi variables, the transferred data sizes are 6 KB and 96 KB with Scibox, compared with 908 MB with FTP transfers of raw data. Another observation is that although network bandwidth is the bottleneck, e.g., for WSU users, Scibox still achieves consistently short latency. This is because for these use cases, due to the relatively small amounts of data transferred, Scibox user experienced latencies are primarily determined by GTS and DR-function execution times. As with

our our experience with other DR-functions, it is clear that for remote users, filtered data sizes are a significant determinant of the performance benefits obtained from using Scibox. Note that for ease of experimentation, our results are obtained with a small GTS data sets, running five particles per cell, resulting in a total file size of 908 MB. Realistic GTS runs typically use 100 particles per cell, resulting in more than a Terabyte of data. For analytics using the phi and $zion$ variables, with those file sizes, Scibox gains additional benefits due to the constant small output size of DR-functions.

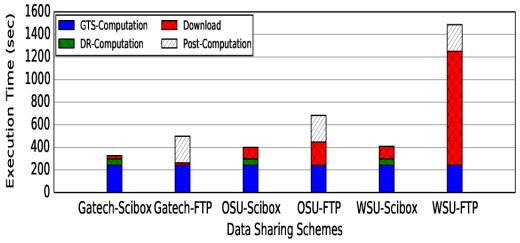

Fig. 12. Comparison of user experienced latency in the GTS experiment.

C. Combustion Workloads

Students in the Aerospace Combustion Laboratory at Georgia Tech need to share thousands of images captured by high-speed cameras in a locally owned experimental combustion facility. The raw data image is not of interest. Instead, students must apply both spatial and time domain Fourier transforms to process images before inspecting them for interesting scientific content. For this case, we develop a DR-function **(ImageName, *DR*8, FFT)** to expedite image processing. In this experiment, 10,000 512×512 12-bit double framed images (about 1.5 MB per image) are shared by an increasingly larger number consumers in a user group. With Scibox, only the transformed images are transferred to the consumers via the cloud. In the stock system, the raw images are transferred and FFT transforms are run on students' local computers. When executing the FFT in Scibox at the producer side, our multi-core servers support high concurrency by load balancing image numbers across parallel transform processes. We increase such parallelism from 1 to 32 and the number of consumers from 1 to 16. Compared to the stock system, we present the performance speedup of experienced latency of Scibox users in Figure 13. In general, the more processes carry out source-based image transformation, the higher the speedup, in part because of the CPU performance disparity between the HPC cluster and students' laptop or desktop computers, with an average speedup of 1.56x. Another observation is that as more consumers share the images, the performance of Scibox benefit is reduced. This is because other system bottleneck compromise its performance advantages, including resource contention in the store's memory buffer cache and for network bandwidth. Our future work will consider in-cloud automated system scale-up for Scibox to deal with its internal performance limitations.

D. Software Overhead Analysis

1) Swift based Object Store: This section first carries out a set of experiments to evaluate the software overhead of the

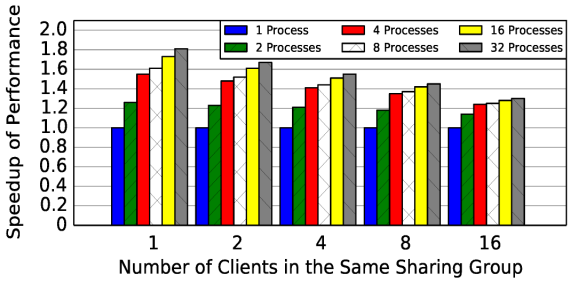

Fig. 13. Comparison of user experienced latency in the combustion image processing experiment.

OpenStack Swift Object Store (a representative open-source solution of cloud storage). We break down the software stack's PUT/GET operations in Swift into four categories: (1) Authorization and Container Checking, (2) Disk Read/Write, (3) Data Transfer and Server Processing, and (4) Other Software Overhead. For the Get operation, we also measure the time of Header Transfer, in which the object name and size are included. For each operation, Swift needs to authorize the account and check whether the accessed container exists in its store. The Disk Read/Write represents the time spent accessing data from local disk. Data Transfer and Server Processing includes the time on the network and for Swift server processing. Experiments measure the time breakdown of every component in the Swift software stack, as a single local producer writes data to the object store using a PUT operation, or as a consumer reads data using the GET operation, increasing object sizes from 1 MB to 128 MB. As shown in Figure 14, when accessing larger objects, the percentage of execution time on Data Transfer and Server Processing increases, which indicates that we are constrained by networking bandwidth. Even in our well-connected local environment, this indicates that it will not typically be useful to transfer entire raw scientific data sets into cloud storage. Instead, a filtering-based solution like Scibox should be provided for scientific data sharing.

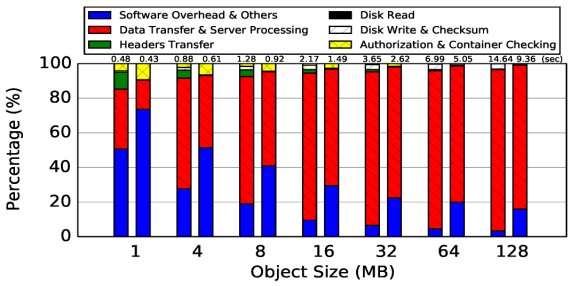

Fig. 14. Running time breakdown of components in the Swift software stack (Left: Get operation, Right: Put operation).

2) System DR-functions: Scibox provides eight DR-functions for users to filter out unwanted data segments and/or to transform the original data sets for subsequent shared use. We first evaluate these DR-functions with the synthetic workloads, to obtain the request execution times when a DR-function is specified in some consumer's requests. Experiments increase the sizes of the variables for uploading, from 64 MB to 1.5 GB. For $DR4$, the range of requested data is also increased, from 1 KB to 128 MB.

In Figure 15, the execution time of $DR1$, $DR2$, $DR3$

is linearly related to the input variable size, since in the processing of each corresponding request, the entire data set needs to be examined to find the max, min, and mean values in the set. In contrast, the execution time of the $DR4$ functions is not affected by the input variable size; its execution time is only determined by data upload time. Therefore, when we increase the range of requested data, its execution time increases accordingly. For other DR functions, we get the similar trends. Due to the space limitation, their results are not included in the paper.

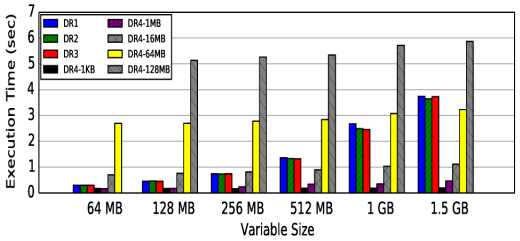

Fig. 15. Execution times of *DR*-Functions

In addition to execution time, the time breakdown of processing each DR-function is presented in Figure 16, including a $DR8$ for which calculating the average of row values of a matrix is used as an example. Total execution time has four part: Get Request which downloads the DR-function from cloud, Request Processing which applies the DR-functions to input variables and merges the their outputs, Put Result which uploads the filtered data in objects to cloud storage, and Others including other software overhead, e.g., object management. As shown in the figure, $DR1$, $DR2$, and $DR3$ have a significant portion of request processing time, in contrast to $DR4$ whose execution times for put results are much longer than the other components. Another observation is that the execution overhead of the CoD-generated $DR8$ type functions is similar to that of the system's DR-functions implemented in C and compiled with the native C compiler.

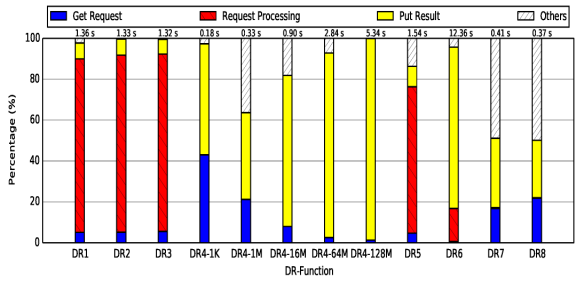

Fig. 16. *DR*-Function execution time breakdown.

VIII. Conclusion

Scibox is a scientific data sharing infrastructure able to operate across both public or private cloud stores. While inspired by commercial systems like Dropbox, Scibox extends its concepts to specifically address the needs of data sharing for science end users. New functionality offered by Scibox exploits the structured nature of science data by sharing metadata across end users before actual data is moved and by then permitting end users to leverage such data to control the amounts of data moved by leveraging users formulate D(ata)R(eduction)-functions.

Experimental results obtained on both wide area and on-campus infrastructures with synthetic benchmarks, with the I/O pipelines used by high end scientific applications, and with workloads derived from analytics actions applied to outputs from scientific instruments. Results demonstrate improved end-to-end performance for data moved from sources to sinks, across the cloud, compared to state-of-the-art approaches. For both remote and on-campus Scibox users, end-to-end data processing time can be reduced on average by 4.08x and 1.56x respectively.

Acknowledgments

This was funded in part by the Cloud Computing Intel Science and Technology Center (Cloud ISTC) and by the Department of Energy under the SDAV SciDac for Data Analytics and Visualization.

References

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: Scalable data staging services for petascale applications," in *HPDC'09*, Munich, Germany, 2009.
[2] SciDAC Review, "Energy science with digital combustors," http://www.scidacreview.org/0602/html/combustion.html.
[3] B. Wilde, C. W. Foley, A. Marshall, and J. M. Quinlan, "A roadmap to advance data processing capability in experimental combustion research via high performance computing," *Technical Report*, 2013.
[4] Microsoft Research, "Experience worldwide telescope," http://www.worldwidetelescope.org/Home.aspx.
[5] "Enabling Grids for E-science," http://www.eu-egee.org/.
[6] "Energe sciences network," http://www.es.net/.
[7] Dropbox Inc., "Dropbox," https://www.dropbox.com/.
[8] Google Inc., "Google Drive," https://drive.google.com/.
[9] Apple Inc., "iCloud," https://www.icloud.com/.
[10] J. Lofstead, M. Polte, G. Gibson, S. A. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu, "Six degrees of scientific data: Reading patterns for extreme scale science io," in *HPDC'11*, San Jose, California, 2011.
[11] K. Madduri, K. Ibrahim, S. Williams, E. IM, S. Ethier, J. Shalf, and L. Oliker, "Gyrokinetic toroidal simulations on leading multi- and manycore hpc systems," in *SC'11*, 2011.
[12] "Lammps molecular dynamics simulator," http://lammps.sandia.gov/.
[13] The HDF Group and the Board of Trustees of the University of Illinois, "HDF5," http://www.hdfgroup.org/HDF5/doc/.
[14] N. Podhorszki, Q. Liu, J.Logan, H. Abbasi, J. Y. Choi, and S. Klasky, "Adios," *ADIOS 1.4.0 Developer's Manual*, 2012.
[15] Amazon Web Services, LLC, "Amazon simple storage service," *Developer Guide*, 2006.
[16] San Diego Supuercomputer Center, "Sdsc cloud storage services," http://cloud.sdsc.edu/hp/index.php.
[17] Google, "Google Drive Storage Plan," https://support.google.com/drive/bin/answer.py?hl=en&answer=2375123.
[18] Dropbox, "Dropbox Team Pricing," https://www.dropbox.com/teams/pricing.
[19] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped gridftp framework and server," in *SC'05*, Seattle, Washington, USA, 2005.
[20] Globus online, "Reliable, high-performance, secure file transfer," https://www.globusonline.org, 2013.
[21] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky, "Just in time: Adding value to the io pipelings of high performance applications with jitstaging," in *HPDC'11*, San Jose, California, 2011.
[22] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, Aug. 2010.
[23] J. Chou, K. Wu, O. Rubel, M. Howison, J. Qiang, Prabhat, B. Austin, E. W. Bethel, R. D. Ryne, and A. Shoshani, "Parallel index and query for large scale data analysis," in *SC'11*, Seattle, Washington, USA, 2011.
[24] M. Stonebraker, J. Becla, D. Dewitt, K. Lim, D. Maier, O. Ratzesberger, and S. B. Zonik, "Requirements for science data bases and scidb," in *Conference on Innovative Data Systems Research*, 2009.
[25] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: Opportunities and challenges at exascale," in *DEBS'09*, 2009.
[26] G. Eisenhauer, M. Wolf, H. Abbasi, S. Klasky, and K. Schwan, "A type system for high performance communication and computation," in *Proceedings of the Workshop on D3Science associated with e-Science11*, Stockholm, Sweden, 2011.
[27] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, *High Performance Event Communication*. CRC Press, 2010, ch. 9.
[28] OpenStack LLC, "Openstack install and deploy manual," 2012.
[29] H. Yoon, A. Gavrilovska, K. Schwan, and J. Donahue, "Interactive use of cloud services: Amazon sqs and s3," in *CCGrid'12*, 2012.