Shock: Active Storage for Multicloud Streaming Data Analysis

Jared Bischof,* Andreas Wilke,* Wolfgang Gerlach,* Travis Harrison,* Tobias Paczian,*
Wei Tang,† Jared Wilkening,‡ Narayan Desai,§ and Folker Meyer*
*Argonne National Laboratory, Argonne, IL, USA
University of Chicago, Chicago, IL, USA
†Google, Inc. USA
‡ Dramafever, Inc., New York, USA
§Ericsson, San Jose, CA, USA
*{jbischof,wilke,wgerlach,teharrison,paczian,folker}@mcs.anl.gov
†weitang@google.com
‡jared.wilkening@gmail.com
§narayan.desai@ericsson.com

***Abstract*—Access to data plays a major role in designing and performing efficient data computation and analyses in a distributed environment. Existing approaches access data via a variety of methods and offer various benefits and drawbacks based on the use case. Our original use case was the computational analysis of environmental sequence data, or metagenomics. Unlike other workflows that often reduce the dataset size dramatically within the first few processing steps, owing to biologially-motivated data compression. Metagenomic data compresses poorly, and so metagenomic workflows add to the size of the data set along the processing pipeline. Thus, wide-area, high-throughput access to the data is essential.**

To address this problem, we developed Shock, a data store for files, their associated metadata, and indexes that allow Shock to provide different views into the data. Shock comprises three major components: a web service that provides a RESTful API, backend data storage for files, and storage for object metadata. Shock has proven to be a stable data store for MG-RAST, an application that served over 40,000 users in 2014 on a server that houses more than 3 million data objects. Moreover, Shock provides both subselection and high-performance file transfer capabilities that serve most usages.

***Index Terms*—bioinformatics, metagenomics, active object store, distributed wide-area computing**

I. Introduction

Access to data plays a major role in efficient data computation and analyses [1]. Existing approaches access data via a variety of methods including local file systems, network-attached persistent block-level storage volumes. Each system offers benefits and drawbacks based on the use case. For example, object-based storage systems provide the needed scalability; but they mostly treat the data as a blob (binary large objects), which is not ideal for splitting large data objects into smaller subunits for parallel computations.

Our study was motivated by a use case involving the computational analysis of environmental sequence data (i.e., metagenomics) [2]. Metagenomics differs from other scientific—and notably bioinformatics—workflows. Whereas other workflows often reduce the size of the data sets dramatically within the first few processing steps [3], metagenomics adds to the size of the data set along the processing pipeline. Moreover, metagenomics often involves decisions that cannot be made until analysis time.

Our particular use case involved the MG-RAST metagenomic analysis portal and pipeline [4]. MG-RAST users upload data via either a web portal or an API [5], and the automated processing pipeline transforms the data into a common file format before producing annotations on the sequences in those files. Some pipeline parameters can be tuned by the users at submission time, and individual work units are then distributed to a pool of clients for processing and analyzing the data. When these computations have completed, the results are stored on the MG-RAST servers for users to download and/or analyze.

When we operated distributed file systems such as NFS or Lustre with MG-RAST, we failed to achieve the required wide-area, high-throughput access to the data. Our solution was to develop Shock—a data store for files, their associated metadata, and indexes that allow users different views into the data—that can efficiently scale data access in a distributed environment.

First, we discuss the requirements of our use case. We then describe the design and implementation of Shock, followed by our results using Shock on MG-RAST. We conclude with a summary and a brief look at future work.

II. Use Case Requirements

Our use case required a data storage solution that could provide access to clients across wide-area networks, administrative domains, and system architectures, without the installation of a specific file system, network, or network access reconfiguration or reformatting. Ideally, this would allow others to dynamically contribute compute clients to our infrastructure while minimizing the amount of overhead needed to do so. We required a data store that could deliver data at high speeds to at least hundreds of compute clients

across remote locations in parallel, with little load on the data server or the clients. We wanted the delivery speed also to be independent of the programming languages being used to perform the computation. Additionally, we wanted the ability to index our data objects in order to serve subsets of the data to multiple compute nodes and/or to perform analysis operations with varying index-defined subsets. And, we wanted users to be able to implement and extend the library of pluggable file parsers. For example, one could imagine using a parser to index all the sequences in a file, but using a second parser to index all of the sequence clusters in the same file.

For efficiency of resources, we desired that our data storage avoid data duplication. And, we wanted to be able to encapsulate all our metadata in this storage engine and make our data searchable by system- and user-defined metadata. Furthermore, we desired a solution where our data not only could be stored for computation but also could serve as a long-term backend data store for our production services. Thus, the data had to be individually addressable without namespace collisions between the numerous temporary objects/files produced during a workflow and the data that is stored long term.

We examined a number of existing technologies for these and other desired features. The results are listed in Table 1. As shown in the table, none of the existing tools met all our requirements. Thus, we sought to create an open source object storage application with features that complemented the existing solutions but also enabled users to plug in subsetting engines specific to their use case, in order to further increase I/O performance in a parallel and distributed compute environment.

TABLE I
SURVEY OF EXISTING TECHNOLOGIES FOR OUR DESIRED FEATURE SET.

	iRods	S3	NFS	Lustre	Gridftp	HDFS
AWAN	+	+	-		+	-
AAB	-	+	-	–	*	-
FCD	-	+	-	+	++	-
INDEX	-	-	-	+	limited	-
Pluggable	-	-	-	-	-	+
ADD	+	-	-	-	+	-
SEARCH	+	-	-	na	-	-
LTS	+	+	na	na	+	?
ZC	-	+	–	–	-	–

AWAN=across wide area networks, AAB=across administrative boundaries, FCD=fast content delivery, INDEX=index data objects and subsets, PLUGGABLE=extend set of pluggable file parsers ADD=avoid data redundancy, SEARCH=searchable metadata, LTS=long-term storage, ZC=zero config

III. SHOCK DESIGN AND IMPLEMENTATION

To meet our needs, we developed Shock, an object-based data management system for biological data and metadata management.

A. Design

The biological data itself should be stored in an efficient backend file system or data store. The metadata can include not just system information about file size, date of creation, and permissions but also complex metadata objects that users can define (a predefined schema is not required). Both system metadata and user metadata are searchable in Shock, enabling users easily to identify their data objects. Allowing for more extensive metadata can help users organize their data objects and inform users of data content and origin. In our target application area, bioinformatics, data reuse is critical as analysis costs increase [6]. Reusability requires comprehensive metadata describing the provenance of original data sets, as well as computational provenance of downstream analysis products. Shock provides consistent and comprehensive provenance storage and query capabilities on this metadata.

Shock has a built-in, optimized, and extendable architecture for index-based subsetting. Users can use the indexing functions that come with Shock or create indexers. Creating subsets from an indexed data file allows users to easily divide their data into subsets for parallel computation and then have Shock merge the results. These features reduce data transfer overhead where clients require only a portion of an input data object for a computation.

Shock provides clients with data access without having to configure access to remote filesystems (and deal with all the accompanying issues) in a wide-area, third-party setting. Shock performs well in a distributed computing environment and serves as a sweet spot between the typical parallel file systems used in HPC environments and distributed object stores. Shock provides parallel streaming of data objects (similar to Amazon's S3) but also provides the tools for creating and storing smart subsets of the data. Each Shock object receives a universally unique identifier, and data is write-once (i.e., once data is uploaded to Shock, it cannot be modified), thus ensuring data coherency.

While filesystems such as Lustre provide server-side seek() operations allowing efficient client-side random access, Shock implements even more computation inside the storage system. This approach can save on more expensive I/O operations. First, Shock computes a checksum on the data uploaded into Shock so that one can verify the successful upload/download of data to and from Shock. Second, upon user request, Shock will build indexes on previously uploaded data object, which can make the data more readily available via subsetting. Third, Shock reduces the need for data duplication by providing the ability to create virtual "copies" of data objects (in actuality, just pointers to a single data object, although this is transparent to the user). Fourth, Shock filters can be used to dynamically reformat data objects on the fly by transforming data that is requested, as it is being downloaded. Fifth, Shock allows for upload followed by subsequent decompression of compressed file types as well as downloading of a file in either compressed or uncompressed format.

Using the AWE [7] compute platform, which integrates with Shock, users can parallelize and scale their applications on distributed compute resources across remote locations. All the uploaded data and all subsequent data products are stored on a Shock server. Each AWE work unit downloads input data objects from Shock, performs some computation, and then outputs data objects back to Shock. Some of the large data

objects that are frequently required for computation are cached on the clients to reduce I/O overhead.

B. Implementation

Architecture: Shock comprises three major components: a web service that provides a REST API, used for all interactions with Shock; a backend data storage that handles the file retrieval, update of metadata, and creation of indexes; and MongoDB, which stores metadata, such as file attributes and properties (ACLs, file size, file checksum, etc.) and user-defined metadata (see Figure 1). Shock allows for complex search queries and supports indexing of metadata fields, thus enabling rapid queries of Shock objects (called nodes) by universally unique identifier, creation date, and object permissions. Additional indexes on node metadata can be specified in Shock to suit different applications.

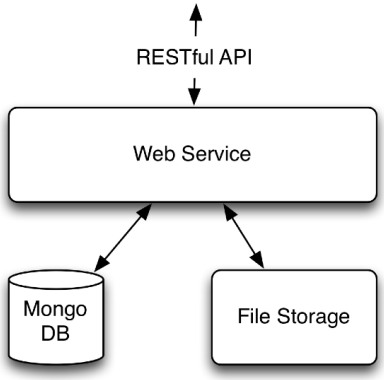

Fig. 1. Design overview of the SHOCK server.

The Shock node data file and indexes on the data itself are stored to disk on the posix file system underlying Shock. Additionally, a BSON file is saved to each node's data directory as a replicate for the JSON document stored in a MongoDB for that node. This file gets updated by Shock each time a modification is made to the node's metadata in MongoDB. Thus, if the MongoDB backend for Shock were lost or corrupted, the data could be recovered from disk by loading these BSON files into MongoDB. Currently Shock supports standard posix file systems; it can be built on top of any parallel or distributed file system and could potentially be extended to run on top of an object store such as Amazon S3. Shock also provides access control via a standard ACL (access control list)-based security model. It can be configured to use external OAuth 2.0 providers (e.g., Globus [8]), and users can share their data with others. Shock is implemented in the Go programming language [9], which provides lightweight threading for performing tasks in parallel and contains extensive network communication libraries suitable for this application.

Shock is an open-source project under a BSD style license maintained at `github.com/MG-RAST/Shock`.

Data records and subsets: The creation of subsets of the data suitable for processing can be handled as a preprocessing step prior to one or many workflow steps; however, if done client side (the normal convention), it will cause significant I/O overhead for the entire data set to be transferred to clients with sufficient local disk available for subsetting. Additionally, results computed on diverse nodes need to be integrated, again requiring significant local storage. In addition to mentioning the resource requirements, ad hoc scripting for subsetting and reintegration is error prone. For this purpose, we implemented the notion of a record in Shock to refer to a single item within a data object.

Within each node's data directory is a subdirectory for storing indexes to the data itself, for providing fast access to structured sections (i.e., records) within the data. Figure 2a depicts how entries in an index file point to a section of the data file. Figure 2b indicates how a chunk-record index is designed to divide contiguous groups of records into similarly sized chunks. This was important for attempting to equally distribute a data set (and hence the computational workload) across many clients in a distributed computing environment. Other supported record types include lines in a text file and lines grouped by column values. Additionally, a list of records in a Shock node can be saved in Shock as a subset for later reuse.

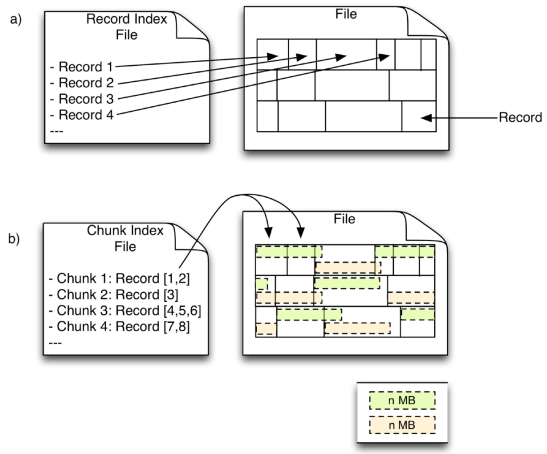

Fig. 2. Graphical depiction of Shock node indexes that reference Shock node data files and provide fast access to regions (i.e., subsets) of the data file.

API: The Shock API is implemented in the Representational State Transfer (REST) style [10]. The API has three resources. The node resource is where users can upload, retrieve, and delete data objects and metadata on the objects. When a new data object is uploaded, a universally unique identifier (uuid) is returned for the newly created Shock node. This makes data objects uniquely addressable across datasets and makes data integration across Shock servers possible. Operations within the node resource allow users to modify ACLs and indexes on the data. The preauth resource is used to provide one-time-use public URL access to data that is otherwise private and requires authentication. The wiki resource is where the Shock API documentation is available for convenience after

TABLE II
SUMMARY OF NODE TYPES

Name	Description
basic node	single file
virtual node	multiple files as one
parts node	parallel creation of a single file in parts
copy node	copy on delete
subset node	subset of a parent node

deployment of a Shock server.

Shock has five node types (see Table 2). A basic node stores a single data file as its contents. A virtual node contains multiple nodes and automatically streams the data for those child nodes as one contiguous file. A parts node is a single node that can be written in multiple pieces. The number of parts in a parts node can either be defined upon creation of the node or declared to be "unknown" . If the number of parts is defined, the node will automatically become immutable once all parts have been uploaded. However, if the number of parts was defined as "unknown", a separate call to "close" the node must be performed to merge the parts that have been uploaded and make the node immutable. Combining the features of node subsetting for download and parts nodes for uploading allows parallel tasks to be split for computation on multiple clients and the results to be merged into a single output node.

The fourth node type is a copy node. This is created by copying an existing node in Shock. When a user requests a copy of an existing node, the underlying data is not copied; rather, a link to the data in the parent node is created in the new node. If the original node is deleted, the data will be moved to a new location, and no information will be lost. Thus, copying a node does not require duplication of the underlying data object. All this is handled by the Shock server thus; the difference between a copy node and a basic node is transparent to the user.

Shock also can create subset nodes. These nodes contain information that represents a subset of the parent node's data. Using any supported record type, one can create a subset node that contains a virtual file representing a subset of the data in the parent node. This is applicable for workflows (especially in bioinformatics) where data goes through several steps of quality filtering and/or sampling and does not necessarily need to be duplicated on disk. For tasks within a workflow for which a filtered dataset is the output, Shock enables users to create a subset node of the parent (input) node by uploading a list of records. For distributed computation, this can have a significant impact on I/O as well.

Metadata and query language: With the ability to store a large number of data objects, users also need a way to query and identify objects in Shock by their metadata with an efficient query language. Metadata in Shock is stored in MongoDB and thus takes the form of a JSON document. This allows users to upload and store complex metadata per node. The supported query language includes the ability to query by Shock-generated node metadata (node ID, file size, file checksum, node type, node creation time, and node modification time) and user-uploaded node metadata (anything stored in the uploaded JSON document). The Shock-generated node metadata is indexed in MongoDB to enhance the performance when querying those fields. Fields within the user-uploaded node metadata can be indexed in MongoDB at the administrative level by adding a list of fields to index to the Shock configuration file at runtime. In addition to providing the query capability for retrieving metadata fields equal to a query value, Shock allows for query by range, string queries with wildcards, and not operators.

Node metadata also can be used to store information about where data came from and how it was derived. Moreover, each Shock node has a separate field called "linkages" where a user can store information about nodes that are linked (parent, child, etc.) to the node object. Linkages form the basis of the provenance chain together with the metadata. Node metadata can also be a location to store descriptions and attributes about the contents of the data.

While Shock can be run without any authentication requirements, the ACLs allow for management of data access permissions. The node ACLs include an owner, a list of users with read permissions, a list of users with write permissions, and a list of users with delete permissions. The Shock administrator and the node owner automatically have read, write, and delete permissions on a node. Only the Shock administrator and the node owner can change permissions on a node by adding or removing a user from an ACL list (this includes changing ownership of a node to another user). Additionally, ACLs can be made "public", thereby granting access to a node without authentication.

IV. RESULTS AND EVALUATION

Shock was initially designed to facilitate the annotation and analysis of large scale metagenomic sequence data within MG-RAST. The requirements of this system included the ability to store and retrieve thousands of submitted sequence files (and millions of the derived files, also stored in Shock), while also being able to serve subsets of these files in parallel to compute clients without blocking on individual requests. Shock has been used as the primary data store (both for serving data to compute clients and as a long-term store) in MG-RAST for more than 18 months now. Figure 3 shows the speeds of uploads and downloads to and from Shock in our production system. The average data transfer speed to and from this server is roughly 100 MB/s. We are consistently operating over 300 AWE clients that are pushing and pulling data to and from a single Shock server while also serving a portion of this data to the scientific community through our production website and API. Note that the figure shows a distinct subset of data objects that take longer to upload to Shock, averaging roughly 1 MB/s. These represent the subset nodes produced during the execution of the MG-RAST pipeline. These nodes take longer to create because they require some server-side computation to store the indices corresponding to these data subsets. Nevertheless, they save a significant portion of disk space on

our Shock server (roughly 15%). Future developments could include optimizing the creation time for subset nodes.

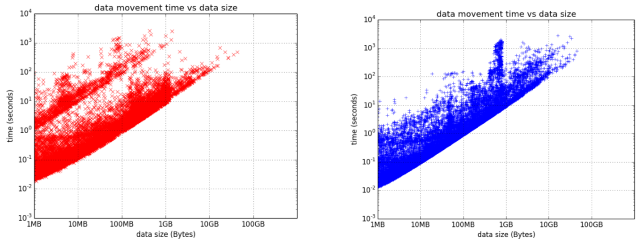

Fig. 3. Upload (red, on the left) and download (blue, on the right) speeds to the MG-RAST Shock server displayed for different data set sizes. This plot represents the load on the production SHOCK server for the MG-RAST system in the period March 1–13, 2015. Some of the upload operations are slower because their creation (subset nodes) requires some computation.

V. Summary and Discussion

Shock has proven to be a stable data store for MG-RAST, an application that served over 40,000 users in 2014 on a server that houses more than 3 million data objects. Combining Shock and AWE, we have provided an efficient model for wide-area workflow execution. Analysis computations are a primary use case for the APIs provided by Shock. These computations have a broad range of workloads: some with a high computation to input data ratio, others with a relatively low ratio. Shock provides both subselection and high-performance file transfer capabilities that serve both usages and anything in between.

Over time we have upgraded from five NFS servers to a single Shock server that serves data in production at a peak rate of nearly 10 Gbit/sec. Several end users for the MG-RAST system have already installed virtual machines with Shock data access on computational systems. We have outlined an API for efficient workflow execution across wide-area networks in metagenomics. This could serve as the basis for a discussion about an efficient wide-area data access API for bioinformatics data in general. This model may not be suitable for all workflow types; but where it is applicable, Shock can be utilized for server-side provisioning and integration of data.

Future goals include extending the Shock application to plug into different backend data storage systems including HDFS and Amazon S3. Extending the application in this manner would make Shock's features appealing to a larger audience. Likewise, by engaging further with the bioinformatics and other big data communities, we hope to extend the abilities of Shock for indexing a wide array of common file types. Furthermore, making the Shock application federated would allow scientists from various locations to share and integrate data more freely and easily, while still maintaining access control and limiting data duplication. The most obvious path appears to be placing objects in a namespace dependent on where they are located; the namespace would need to be globally maintained and would allow objects to be individually addressable across all Shock servers.

Acknowledgment

The authors would like to thank Gail Pieper for her invaluable help with editing the manuscript.

This work was supported in part by the NIH award U01HG006537 "OSDF: Support infrastructure for NextGen sequence storage, analysis, and management", and U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research DE-AC02-06CH11357 as part of "Resource Aware Intelligent Network Services (RAINS)". Computing for this work was supported in part by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

References

[1] Department of Energy, Office of Science, ASCR, "Storage systems and input/output to support extreme scale science: Report of the DOE workshops on storage systems and input/output," in *Report of the DOE workshops on storage systems and input/output. Rockville, Maryland. Dec 8-11, 2014*, 2014.

[2] T. Thomas, J. Gilbert, and F. Meyer, "Metagenomics-a guide from sampling to data analysis," *Microb Inform Exp*, vol. 2, no. 3, pp. 1–12, 2012.

[3] N. Desai, D. Antonopoulos, J. A. Gilbert, E. M. Glass, and F. Meyer, "From genomics to metagenomics," *Current opinion in biotechnology*, vol. 23, no. 1, pp. 72–76, 2012.

[4] F. Meyer, D. Paarmann, M. D'Souza, R. Olson, E. Glass, M. Kubal, T. Paczian, A. Rodriguez, R. Stevens, A. Wilke, J. Wilkening, and R. Edwards, "The metagenomics RAST server - a public resource for the automatic phylogenetic and functional analysis of metagenomes," *BMC Bioinformatics*, vol. 9, no. 1, p. 386, 2008. [Online]. Available: http://www.biomedcentral.com/1471-2105/9/386

[5] A. Wilke, J. Bischof, T. Harrison, B. Thomas, M. D'Souza, W. Gerlach, H. Matthews, T. Paczian, J. Wilkening, E. M. Glass, N. Desai, and F. Meyer, "A RESTful API for Accessing Microbial Community Data for MG-RAST," *PLoS Computational Biology*, vol. 11, no. 1, 2015.

[6] National Human Genome Research Institute (NHGRI), "DNA Sequencing Costs – Data from the NHGRI Genome Sequencing Program (GSP)," http://www.genome.gov/sequencingcosts/, 2015, [Online; accessed 19-July-2015].

[7] W. Tang, J. Wilkening, N. Desai, W. Gerlach, A. Wilke, and F. Meyer, "A scalable data analysis platform for metagenomics," *2013 IEEE International Conference on Big Data*, 2013.

[8] Globus, "Research data management simplified," http://www.globus.org, 2015, [Online; accessed 19-July-2015].

[9] "The GO programming language," http://www.golang.org, 2015, [Online; accessed 19-July-2015].

[10] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine., 2000, [Chapter 5: Representational State Transfer (REST)].