

getdata: A Grid Enabled Data Client for Coastal Modeling

Dayong Huang^{1,2}, Gabrielle Allen^{1,2}, Chirag Dekate^{1,2}, Hartmut Kaiser²,
Zhou Lei² and Jon MacLaren²

¹Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA

²Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803, U.S.A.
{dayong, gallen, cdekate, hkaiser, zlei, maclaren}@cct.lsu.edu

Abstract

The SURA Coastal Ocean Observing and Prediction (SCOOP) program is implementing an integrated system of regional observations and computational models to provide improved coastal forecasts. As part of this system, the getdata client was developed to search and transfer data files from the SCOOP archive in a flexible, optimized and extensible manner. The Grid Application Toolkit forms the basis for this tool, providing an abstract interface to various transfer protocols including FTP, HTTP, SCP and GridFTP. The user requirements for getdata posed a number of constraints on the client design and integration with Grid middlewares. This paper discusses the SCOOP use cases and the design and implementation of getdata, along with the challenges of engineering usable tools for heterogeneous environments and diverse communities.

1. Introduction

The SURA Coastal Ocean Observing and Prediction (SCOOP) project [1] is a collaboration between coastal modeling researchers, operational agencies, and computer scientists to integrate data from regional observing systems with advanced computational models for improved real-time coastal forecasts in the southeast United States. The project involves deploying ensembles of different coupled models over different geographical areas in three modes: operationally (24/7), event-driven for incoming hurricanes or tropical storms, and for retrospective analysis of past events. Part of the work flow involves the realtime verification and visualization of predictions.

The SCOOP partners are representative of a growing class of geographically distributed collaborators who are investigating the use of Grid technologies to help share expertise, data, software and resources between multiple institutions. These collaborations, extending across multiple administrative domains, require easy-to-use, reliable and homogeneous tools to perform tasks such as data placement

and model workflow execution.

Current widely accepted Grid infrastructure, such as Globus and Condor, provides various low level components and middleware to integrate remote resources and deploy applications. However, high level tools and services are needed to make this functionality useful for application communities, and address issues such as fault tolerance, reliability and quality of service. This paper describes the design and implementation of such a tool, called *getdata*, that can locate and move coastal data among the SCOOP partners. The tool is built using the Grid Application Toolkit [2] which provides an abstract layer for application oriented Grid capabilities, and as such will be relevant to other communities with similar needs.

Data management is a pressing issue for the coastal modeling and observing community. Data is generated and stored in many different locations, in different formats, and transported over different protocols which are often specific to the community (e.g. OPeNDAP [3], LDM [4]). Scientists working in the SCOOP collaboration have established an advanced Grid-enabled data storage archive service [5], providing essential features such as digestion and retrieval of data via multiple protocols (including LDM, GridFTP and HTTPS), a logical file catalog, and general event-based notification. Various data standards are being developed in the community to facilitate sharing and collaboration [6], but in lieu of an existing formal standard, the SCOOP partners have developed a file-naming convention across the project that encodes enough information to serve as primary metadata.

2. Usecases and Requirements

The SCOOP archive service [5] at LSU currently stores atmospheric, wave and surge datasets, as well as observational data. These datasets are produced by the SCOOP partners, as well as by dozens of different operational entities in the US. The archive currently stores around 100,000 files at any time (representing the last month of data), and

files range in size from a few kilobytes to 250Mb.¹ A file naming convention is used to encode metadata which describes the contents of any data file, in the future a fuller metadata service will be deployed. The following core use-cases describe how data from the archive will be used:

Automated operational coastal model workflows of different hydrodynamic scenarios run continuously. These time constrained workflows require data to be transferred from the archive to the run location.

Ensembles of hurricane models triggered by notifications of predicted tracks from the National Hurricane Center generate wind and pressure fields, which are ingested into the archive, triggering multiple hydrodynamic models needing timely access to the latest datasets in the archive.

Retrospective event models are deployed at will by coastal researchers to provide skill assessment and investigate different algorithms and parameter sets.

Each scenario requires searching for data sets with metadata, and then copying data sets to one or more compute resources. Although a portal interface to the archive has been developed, a command line data client is needed which can be easily used by the scientists as part of their usual work environment, either interactively from a terminal, or in batch mode in a script.

A SCOOP Grid is being deployed with services such as GridFTP, however a key requirement for this data client is that it should be usable on existing machines used by the coastal modelers which are not yet part of the Grid. The machines are located on networks with differing characteristics, from high speed optical networks, through Internet2, to lower bandwidth infrastructures. The data client should be able to automatically adapt to use the most appropriate protocol for available network and data set.

The SCOOP project has relatively simple and homogeneous security needs. The data sets are currently restricted to the SCOOP partners, mainly out of concern that data not be misinterpreted. The data tools should provide read-only access to datasets in the archive and auxiliary data management services such as replica catalogs for restricted users.

The data client must be able to support multiple protocols providing varying levels of qualities of service based on scheduling constraints. In other words, the tool should be able to opportunistically utilize high QoS protocols such as GridFTP, depending on the size of the data sets and provided that the supporting middlewares (Globus) are available on the hosts where the data client is deployed. The users should be able to deploy the tool across wide variety of resources with little effort. Many of the motivating usecases are severely time constrained events requiring the ability to access datasets and retrieve them in

timescales which allow for the entire process of data pre-processing, model initiation and output data delivery to complete within a fixed time. The ability to select different protocols, select number of streams for data download, and other such data delivery optimization parameters would significantly enhance capabilities to complete staging of coastal models. Since the end users of the client are coastal modelers with varying levels of computer expertise, the tool must be easy to deploy and use.

3. Data Management using GAT

Several Grid data replication schemes and high speed data transfer protocols have been developed to handle the needs of modern distributed scientific applications. In the Globus toolkit, the Replication Location Service (RLS) provides logical file abstraction [7], and GridFTP [8] provides a high speed transfer protocol which uses the Grid Security Infrastructure (GSI). These existing tools for data management each have their own application programming interface, which the grid application developer can use to build higher level services.

To take advantage of different Grid middlewares, an application programmer needs to understand how different middleware APIs work. That means the application has to be coded against different underlying Grid middlewares. Grid application developers building higher level services ideally need to take into account a range of existing Grid solutions, and provide support for legacy utilities with a migration path towards the advanced solutions. All this is a potentially overwhelming endeavor.

To avoid deep dependencies on a certain set of grid middleware the getdata tool is built on top of the Grid Application Toolkit (GAT) [2]. The GAT is designed to handle the diversity of Grid middlewares, providing a consistent abstraction layer (the GAT API) to Grid application developers. The GAT's main objective is to provide a single, easy-to-use Grid API, while hiding the complexity and diversity of the actual Grid resources and their middleware layers. The GAT provides abstractions for computers, data, and application instances.

The Grid Application Toolkit is comprised of two parts, a GAT engine and a set of GAT adaptors (Figure 1). The GAT engine exposes a set of APIs to the application programmer, the getdata tool is built against this GAT API. The GAT adaptors are a set of lightweight software elements which provide access to specific Grid capabilities. At run time any calls to GAT APIs are relayed by the GAT engine to a corresponding adaptor and to the grid middleware. The layered GAT architecture and the GAT API has been described in more detail in [2]. By using the GAT we were able to eliminate all direct grid middleware API calls in the getdata tool, delegating the underlying Grid work to

¹An additional archive maintained at TAMU stores 1.4Tb of files which will be integrated with the LSU archive.

the GAT adaptors.

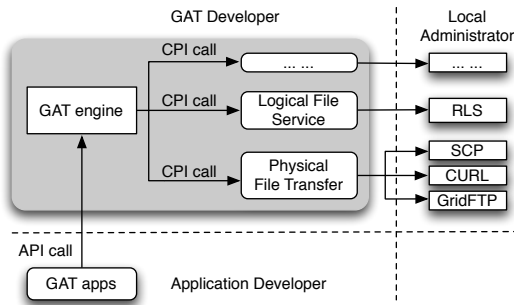


Figure 1. Generic data client using the Grid Application Toolkit

The GAT supports a growing set of Grid data management middlewares. The Globus RLS and Gridlab logical file service are supported as logical file service. For physical data transfer, GridFTP, SRB, SCP, Reliable UDP, and cURL adaptors are available, and a Globus Reliable File Transfer adaptor is being developed. The GAT application programmer only needs to use a standard GAT call, and the GAT engine then handles the actual service calls.

4. getdata: Design and Implementation

The Globus Replica Location Service (RLS) was deployed to provide part of the data management requirements for SCOOP. When files are ingested into the SCOOP archive, logical filename entries are created to correspond with the physical storage location and entered into the RLS. The getdata client is a command-line tool that allows users to locate files using the Replica Location Service and download the file from the archive using different data access protocols. In this section we discuss key issues in the design and implementation of getdata.

4.1. getdata Architecture

When a user initiates the process of browsing or retrieving data from the storage archive, the following series of steps are carried out (see Figure 2):

- R1:** The user formulates a query string based on metadata.
- R2:** The query string (which can contain wildcards) is passed to the logical file server which returns associated Physical File Names (PFNs).
- R3:** The user reviews the provided PFNs, and provides the destination location for any PFNs which should be copied.
- R4:** When used in batch mode, the user provides the query string and the destination location, along with any associated preferences and getdata transfers the file in the background without requiring further user intervention.

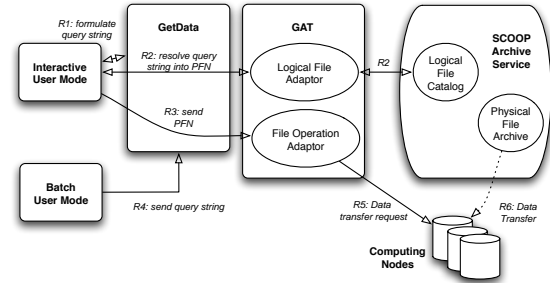


Figure 2. Basic architecture for the getdata archive client

R5: The getdata utility invokes the GAT Engine to load the actual service (adaptor), to perform data transfer to the specified location.

R6: The physical data is downloaded from the storage archive to the destination location.

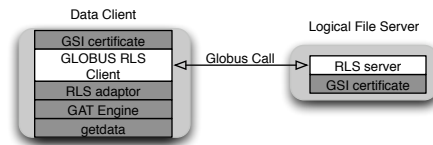


Figure 3. Design of the getdata utility using the GAT for providing abstraction from Grid middleware.

For developing the getdata using GAT (see Figure 3), two sets of GAT APIs for data management are used, namely the logical file service API and the physical file transfer API. The GAT logical file API is an interface which resolves a query string to the appropriate list of physical file names. The GAT physical file transfer API provides various capabilities such as file copy, move and deletion.

When the GAT engine handles logical file service API calls (see part 1 in Figure 4), it determines the set of registered adaptors that provide the requested logical file capabilities. Since RLS is the only logical file adaptor used in our scenario, the GAT engine selects the RLS adaptor, and passes all logical file queries to the underlying RLS middleware.

When the GAT engine encounters a physical file transfer API call (see part 2 in Figure 4), it determines set of registered adaptors that can provide the requested physical file transfer capabilities. There can be multiple adaptors that have registered to provide the physical file transfer capabilities, e.g. the GridFTP, HTTP, SCP adaptors. Currently, when initialized the GAT engine binds to the first adaptor that registers its capabilities. Therefore, during initialization if GridFTP adaptor registers with the GAT engine for providing file transfer capabilities, all ensuing data transfer operation calls are passed on to the GridFTP adaptor.

```

int getPFNs (char * querystring) {
/* initializations */
logical_name = GATLocation_Create(querystring);
logical_file = GATLogicalFile_Create (context, logical_name,
GATLogicalFileMode_Create, NULL);
if (GATLogicalFile_GetLocations (logical_file, &locations)) {
/* use iterator to list all the physical file names from locations */
}
}

int copyfile (char * srclocation, char * destlocation) {
/* initializations */
srcfile = GATFile_Create (context, srclocation, NULL);
if (! GATFile_Copy(srcfile, destlocation, ), GATFileMode_Overwrite))
{ /* err handling */ }
}

```

Figure 4. The GAT Logical File API code that resolves the physical file name for the given logical file names.

The ability to select between multiple protocols is provided because in production environments that involve multiple administrative domains such as in the SCOOP project, homogeneous middleware cannot be guaranteed. The GAT distribution provides several adaptors to handle such diverse environments. In absence of underlying Grid middlewares, the GAT engine attempts to load the next available adaptor that offers similar capabilities. If the GAT engine is unable to find an appropriate adaptor-middleware match then it returns an error message.

4.2. Distribution and RLS wrapper service

The getdata utility, the GAT and the associated adaptors need to be usable in environments where middleware such as Globus is not installed. Achieving a workable distribution of the client posed significant challenges, particularly with the RLS related adaptors, since the adaptor needs to load the RLS libraries at runtime. The dependency of the client on RLS (through the GAT adaptor) was very restrictive, even if the data files could be transferred with HTTP it requires a Globus installation on end user machines.

To overcome this problem we sought to distribute static binaries, including the RLS libraries. This was however not successful since the RLS libraries in turn tried to load other libraries such as *globus_xio* at runtime. A more successful solution was however found. A RLS wrapper web service was engineered to provide intermediary services between the getdata utility and the SCOOP RLS instance, removing the dependency (at least on RLS) at the client sites.

The RLS wrapper service (see Figure 5) was developed using the GAT logical file Capability Provider Interface. The wrapper service on the server side uses the GAT, the dependent Grid middleware and valid GSI credentials for the service to interact with the RLS server using the RLS

adaptor. The getdata client uses a GSOAP logical file adaptor to interact with the wrapper service. The communication between the RLS wrapper web service and the getdata client is encapsulated as GSOAP [9] messages. The wrapper service also makes the security issues easier. The wrapper server now holds the credential to access the RLS, and implements only read access to the RLS server.

The getdata client utility did not need to be modified at all to incorporate the RLS wrapper solution. From a Grid application developer perspective, not having to rewrite application codes and relying on higher level Grid APIs for consistent services to the grid middlewares, produces scalable middleware that can adapt to middleware changes.

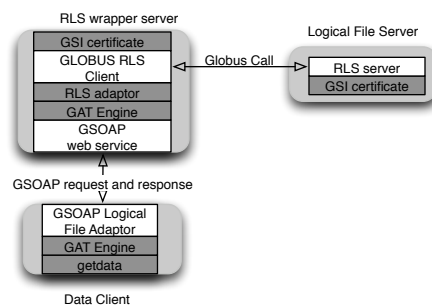


Figure 5. RLS Wrapper service design for relaying queries from the getdata tool to the RLS server instance

The wrapper service uses GAT calls to the logical file service, which is thus not restricted to be Globus RLS, any logical file service that can resolve the query string into physical file names can be used. This provides the flexibility to use other logical file services in the future.

The final solution needs Globus libraries on the server side, a requirement that can be met easily. On the client side, getdata, relies on the GSOAP logical file adaptor instead of the RLS adaptor, and the dependency on RLS libraries is no longer a limiting factor for distribution. End users using getdata to download data from the archive using HTTPS are no longer required to install Globus packages.

4.3. Transfer Optimization with GAT

In practical deployment scenarios, multiple adaptors can provide the same set of capabilities, for example physical file transfer support can be provided by the GridFTP, SCP, cURL or other adaptor. Currently the GAT Engine cycles through the list of adaptors selecting the first adaptor-middleware match that claims to provide the required services. However this behavior could be better engineered to provide adaptive selection of adaptors based on the working environment and conditions.

The GAT adaptor selection mechanism currently does

not support decisions based on optimization criteria, so the adaptor choosing mechanism is built into the `getdata` tool itself to ensure that the required adaptor is used. The GAT engine provides a special API for this, although putting the burden of this low level programming onto the application developer contradicts a main purpose of the GAT. Nevertheless, in this first step, complex service selection use cases need first to be tested before generic solutions are directly implemented in the GAT engine. In the future the adaptor selection intelligence will be implemented as a part of a standard GAT distribution by providing a new set of APIs allowing to incorporate different selection criteria.

Generally protocol selection (and hence adaptor selection) could be based on two sets of criteria:

- **Inferential selection using middleware characteristics:** Middleware characteristics, such as the existence of valid Grid proxies or `ssh-agents`, could be used to shortlist adaptors that provide the requested capabilities. For example, information such as a valid GSI proxy or `ssh-keys` could lead to shortlisting GridFTP and SCP adaptors for data transfers.
- **Metrics based selection:** Metrics such as file size and network performance could be used as criteria for adaptor selection. In environments where high bandwidth networks are available, data transfers could benefit from parallelization, whereas as in low bandwidth networks, parallelisms would increase the overhead.

For inferential selection, `getdata` relies on the adaptors, and tries to execute a file transfer based on a certain adaptor. A success indicates availability of the corresponding protocol. Metrics based selection currently is implemented based only on file size. Network characteristics will be incorporated into the adaptor selection process in the future.

Tuning and optimization of data transfer protocols has been widely studied [10,11]. Results from such studies providing comparative criteria for protocol selection are being incorporated into `getdata`. The current implementation uses the following criteria for selecting between GridFTP, `cURL` and SCP adaptors.

Silberstein et.al. [12] analyzed the effect of file sizes on the performance of local and wide-area GridFTP transfers, finding that files sizes should be at least 10MB for slow wide-area connections and 20MB for fast local-area connections to achieve 90% of optimal performance, and small files do not benefit from multiple streams due to the increased overhead of stream management. Our own measurements (Table 1) comparing GridFTP (no parallel streams, 5000 bytes block size) and HTTP protocols do not contradict these findings.

Based on this knowledge the `getdata` tool currently chooses GridFTP over HTTP (implemented by the `cURL`

| Overall data size (bytes) | GridFTP transfer rate (Mbyte/s) | HTTPS transfer rate (Mbyte/s) |
|---------------------------|---------------------------------|-------------------------------|
| 362808 | 0.130281 | 0.983428 |
| 985752 | 0.246001 | 1.654858 |
| 15081768 | 2.151818 | 2.674489 |
| 40996332 | 3.904769 | 3.144733 |

Table 1. Example average transfer rates for different transfer protocols. In each case 260 transfers from the SCOOP archive to a local machine were made.

adaptor) for files greater than 20MByte, as long as GridFTP is available. Since on high bandwidth networks data transfers benefit from parallelization (see [10]) we additionally use 4 parallel GridFTP streams for large files.

5. Related Work

Distributed data-intensive applications in modern scientific computation require scalable data management components and systems that can span multiple administration domains, multiple storage systems types, and multiple kinds of data access environments. The Globus team proposed a Data Grid infrastructure design in 1999 [13], since when much work has been done at the system and component level.

GriPhyN is a NSF funded project aiming to build a virtual Grid for petascale data management. VDT, Pegasus and Chimera are the major data management systems developed by GriPhyN. Using data provenance, they aim to reuse data products when integrating data into computations. The GriPhyN system has been applied to large data-intensive applications in astronomy, high energy physics and gravitational wave observation.

The Lightweight Data Replicator (LDR) [14] is another important data management middleware developed for the LIGO project. LDR replicates data sets from instruments to the member sites of a Virtual Organization or Data Grid. The goal of LDR is to use the minimum collection of components necessary for fast and secure replication of gravitational wave data. In its present form, LDR only uses a single data transport protocol (GridFTP).

The Storage Resource Broker (SRB) [15] is client-server middleware developed at SDSC. It provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. In conjunction with the Metadata Catalog (MCAT), SRB provides a way to access data sets and resources based on their attributes and/or logical names rather than their names or physical locations. SRB provides a set of command line tools, with `Sget` providing similar capabilities to `getdata`. An GAT adaptor for SRB is available, and the LSU SCOOP

archive will be integrated with SRB in the near future.

Stork represents recent advances in the Grid data management field. It treats data transfer as jobs, and data transfer can be matched to resources and scheduled. Stork provides adaptive, fail-safe, and fully automatic data transfer.

Similar to GAT, the SAGA [16] Research Group at GGF is defining a standard high level API for developers of Grid applications. The SAGA API will be supported by GAT.

6. Conclusion and Future Work

We have described a Grid enabled data client for coastal modeling called getdata which is built using the Grid Application Toolkit (GAT). The generic interfaces of the GAT provides for future extensibility, as new GAT adaptors for logical files services and data movement become available, they can trivially be adopted.

We discussed issues in the deployment and implementation of getdata. While GAT offers application programmer great flexibility to develop Grid applications, local administrators need to realize that GAT is an interface to other middlewares instead of a self-contained bundle of functional components. GAT applications are dependent on the underlying Grid middleware. For non-webservice Grid middlewares, GAT will not be able to offer Grid functionalities without the middleware client software and Grid certificate installed at the client sites. We showed in Section 4.2 that this dependency can be alleviated by adding web service wrappers, such that only the web service client needs to be installed at the client sites. The Grid middleware and Grid certificate is then only required at the wrapper server, reducing the administration work at the client sites.

The ability to choose the right adaptor at run time is essential to the success of GAT. Currently GAT tries each installed adaptor until it succeeds. We have put some effort in improving the choosing scheme, introduced in Section 4.3. In the future we plan to implement more advanced techniques in the data client for further optimization. This work can then be implemented in the GAT Engine.

7. Acknowledgments

This work was carried out through the SURA Coastal Ocean Observing and Prediction (SCOOP) Program, an initiative of the Southeastern Universities Research Association (SURA). Funding for SCOOP is provided by the Office of Naval Research, Award N00014-04-1-0721 and by the National Oceanic and Atmospheric Administrations NOAA Ocean Service, Award NA04NOS4730254. Additional funding was provided by the Center for Computation & Technology at LSU. Zhou Lei acknowledges support from DOE DE-FG02-04ER46136 (UCOMS).

References

- [1] SURA Coastal Ocean Observing Program (SCOOP). http://www1.sura.org/3000/3300_Coastal.html.
- [2] G. Allen, K. Davis, K.N. Dolkas, N.D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzycki, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, and I. Taylor. Enabling applications on the Grid: A GridLab overview. *International Journal of High Performance Computing Applications*, 17:449–466, 2003.
- [3] Open-source Project for a Network Data Access Protocol (OPeNDAP). <http://www.opendap.org/>.
- [4] Unidata Local Data Manager (Unidata). <http://www.unidata.ucar.edu/software/ldm/>.
- [5] Jon MacLaren, Gabrielle Allen, Chirag Dekate, Dayong Huang, Andrei Hutanu, and Chongjie Zhang. Shelter from the Storm: Building a Safe Archive in a Hostile World. In *Proceedings of the The Second International Workshop on Grid Computing and its Application to Data Analysis (GADA'05)*, Agia Napa, Cyprus, 2005. Springer Verlag.
- [6] Marine Metadata Interoperability (MMI). <http://marinemetadata.org/>.
- [7] Ann Chervenak, Naveen Palavalli, Shishir Bharathi, Carl Kesselman, and Robert Schwartzkopf. Performance and scalability of a replica location service. In *13th IEEE International Symposium on High Performance Distributed Computing*, Honolulu, Hawaii, 2004. IEEE Press.
- [8] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. Gridftp: Protocol extensions to FTP for the grid. *Global Grid Forum Recommendation Document GFD.20.*, 2005.
- [9] R. van Engelen and K. Gallivan. The GSOAP toolkit for web services and peer-to-peer computing networks. In *Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid*, pages 128–135, Berlin, Germany, 2002. IEEE Press.
- [10] George Kola, Tevfik Kosar, and Miron Livny. Profiling grid data transfer protocols and servers. *10th International Euro-Par Conference, Pisa, Italy*, 1:452–459, 2004.
- [11] Sudharshan Vazhkudai, Steven Tuecke, and Ian Foster. Replica selection in the globus data grid. *Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid CCGRID 2001*, 1:106–113, 2001.
- [12] M. Silberstein, M. Factor, and D. Lorenz. Dynamo - directory, net archiver and mover. *Proceedings of the third International Workshop on Grid Computing, Baltimore, MD*, 1:256 – 267, 2002.
- [13] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets, 1999.
- [14] Lightweight Data Replicator (LDR). <http://www.lsc-group.phys.uwm.edu/LDR/>.
- [15] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The sdsc storage resource broker. Toronto, Canada, 1998. CASCON.
- [16] SAGA Research Group. <https://forge.gridforum.org/projects/saga-rg>.