

A peer-reviewed version of this preprint was published in PeerJ on 15 January 2018.

[View the peer-reviewed version](https://peerj.com/articles/cs-144) (peerj.com/articles/cs-144), which is the preferred citable publication unless you specifically need to cite this preprint.

Chard K, Dart E, Foster I, Shifflett D, Tuecke S, Williams J. 2018. The Modern Research Data Portal: a design pattern for networked, data-intensive science. PeerJ Computer Science 4:e144
<https://doi.org/10.7717/peerj-cs.144>

1 The Modern Research Data Portal: 2 A Design Pattern for Networked, 3 Data-Intensive Science

4 Kyle Chard¹, Eli Dart², Ian Foster¹, David Shifflett¹, Steven Tuecke¹, and
5 Jason Williams¹

6 ¹The University of Chicago and Argonne National Laboratory

7 ²Energy Sciences Network, Lawrence Berkeley National Laboratory

8 Corresponding author:

9 Ian Foster

10 Email address: foster@anl.gov

11 ABSTRACT

12 We describe best practices for providing convenient, high-speed, secure access to large data via research
13 data portals. We capture these best practices in a new design pattern, the Modern Research Data
14 Portal, that disaggregates the traditional monolithic web-based data portal to achieve orders-of-magnitude
15 increases in data transfer performance, support new deployment architectures that decouple control
16 logic from data storage, and reduce development and operations costs. We introduce the design pattern;
17 explain how it leverages high-performance Science DMZs and cloud-based data management services;
18 review representative examples at research laboratories and universities, including both experimental
19 facilities and supercomputer sites; describe how to leverage Python APIs for authentication, authorization,
20 data transfer, and data sharing; and use coding examples to demonstrate how these APIs can be
21 used to implement a range of research data portal capabilities. Sample code at a companion web site,
22 <https://docs.globus.org/mrdp>, provides application skeletons that readers can adapt to realize
23 their own research data portals.

24 INTRODUCTION

25 The need for scientists to exchange data has led to an explosion over recent decades in the number
26 and variety of **research data portals**: systems that provide remote access to data repositories for such
27 purposes as discovery and distribution of reference data, the upload of new data for analysis and/or
28 integration, and data sharing for collaborative analysis. Most such systems implement variants of a design
29 pattern (Gamma et al., 1994) that we term the **legacy research data portal** (LRDP), in which a web
30 server reads and writes a directly connected data repository in response to client requests.

31 The relative simplicity of this structure has allowed it to persist largely unchanged from the first days
32 of the web. However, its monolithic architecture—in particular, its tight integration of control channel
33 processing (request processing, user authentication) and data channel processing (routing of data to/from
34 remote sources and data repositories) has increasingly become an obstacle to performance, usability, and
35 security, for reasons discussed below.

36 An alternative architecture re-imagines the data portal in a much more scalable and performant form.
37 In what we term here the **modern research data portal** (MRDP) design pattern, portal functionality is
38 decomposed along two distinct but complementary dimensions. First, control channel communications
39 and data channel communications are separated, with the former handled by a web server computer
40 deployed (most often) in the institution's enterprise network and the latter by specialized data servers
41 connected directly to high-speed networks and storage systems. Second, responsibility for managing data
42 transfers, data access, and sometimes also authentication is outsourced to external, often cloud-hosted,
43 services. The design pattern thus defines distinct roles for the web server, which manages who is allowed
44 to do what; data servers, where authorized operations are performed on data; and external services, which
45 orchestrate data access.

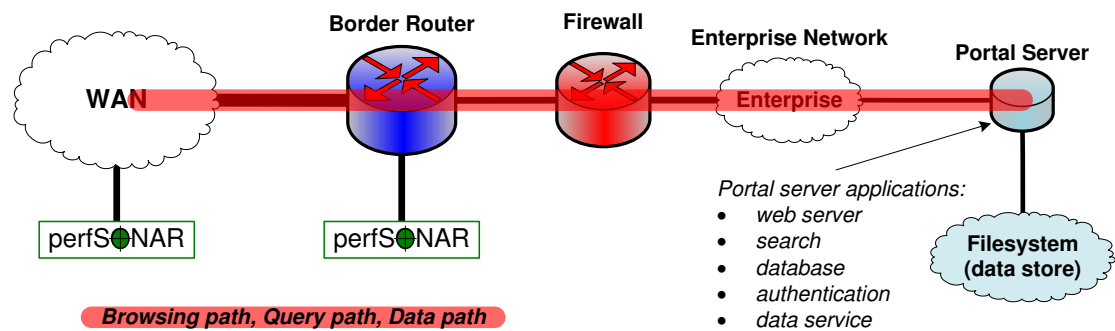


Figure 1. The LRDP architecture. The portal web server both runs all portal services and handles all data objects.

46 In this article, we first define the problems that research data portals address, introduce the legacy
 47 approach, and examine its limitations. We then introduce the MRDP design pattern and describe its real-
 48 ization via the integration of high-performance network architectures and cloud-based data management
 49 and authentication services, for which we use the Science DMZ (Dart et al., 2013) and Globus (Chard
 50 et al., 2014) as canonical examples, respectively. We then outline a reference implementation, also
 51 provided in its entirety on the companion web site, <https://docs.globus.org/mrdp>, that the
 52 reader can study—and, if they so desire, deploy and adapt to build their own MRDP. We also review
 53 various deployments to show how the MRDP approach has been applied in practice: examples like the
 54 National Center for Atmospheric Research’s Research Data Archive, which provides for high-speed data
 55 delivery to thousands of geoscientists; the Sanger Imputation Service, which provides for online analysis
 56 of user-provided genomic data; the Globus data publication service, which provides for interactive data
 57 publication and discovery; and the DMagic data sharing system for data distribution from light sources.
 58 We conclude with a discussion of related technologies and summary.

59 THE RESEARCH DATA PORTAL

60 The exchange of data among researchers is a fundamental activity in science (Borgman, 2012; Hey et al.,
 61 2009; Tenopir et al., 2011), and the use of computer networks for that purpose dates back to the earliest
 62 days of the Internet. Indeed, it was the need for convenient data exchange that drove Tim Berners-Lee
 63 to invent the web in 1989 (Berners-Lee, 1989). As is well known, he defined an architecture of extreme
 64 simplicity. A computer with access to both a data repository and the Internet runs a web server applica-
 65 tion that performs upload and download operations on its local data repository based on client requests. Users
 66 issue such requests using a client program—typically a web browser, which provides a graphical user
 67 interface (GUI). A uniform naming scheme for data objects makes it easy to share names, for example by
 68 embedding them in specially formatted documents (web pages).

69 Web technologies have since evolved tremendously. For example, web servers can now support
 70 client authentication and authorization, link to databases for efficient navigation of large repositories, and
 71 support Common Gateway Interface (CGI) access to server-side computation. Web browsers run powerful
 72 Javascript and support asynchronous requests to remote servers.

73 Yet as shown in Figure 1, most systems used to exchange research data today are not so different from
 74 that first web server. In particular, a single server handles request processing, data access, authentication,
 75 and other functions. It is the simple and monolithic architecture that characterizes this **legacy research**
 76 **data portal** (LRDP) design pattern that has allowed its widespread application and its adaptation to many
 77 purposes. Under names such as portals (Russell et al., 2001), science gateways (Wilkins-Diehr et al., 2008;
 78 Lawrence et al., 2015), and hubs (Klimeck et al., 2008; McLennan and Kennell, 2010), LRDP instances
 79 variously support access to small and large scientific data collections, data publication to community
 80 repositories, online data analysis and simulation, and other scientific workflows concerned with managing
 81 the flow of scientific data between remote clients and a central server.

82 A confluence of three factors has now rendered the monolithic LRDP architecture increasingly
 83 problematic for its intended purpose, as we now discuss.

84 The first concerns **performance**. Both data volumes and network speeds have grown tremendously

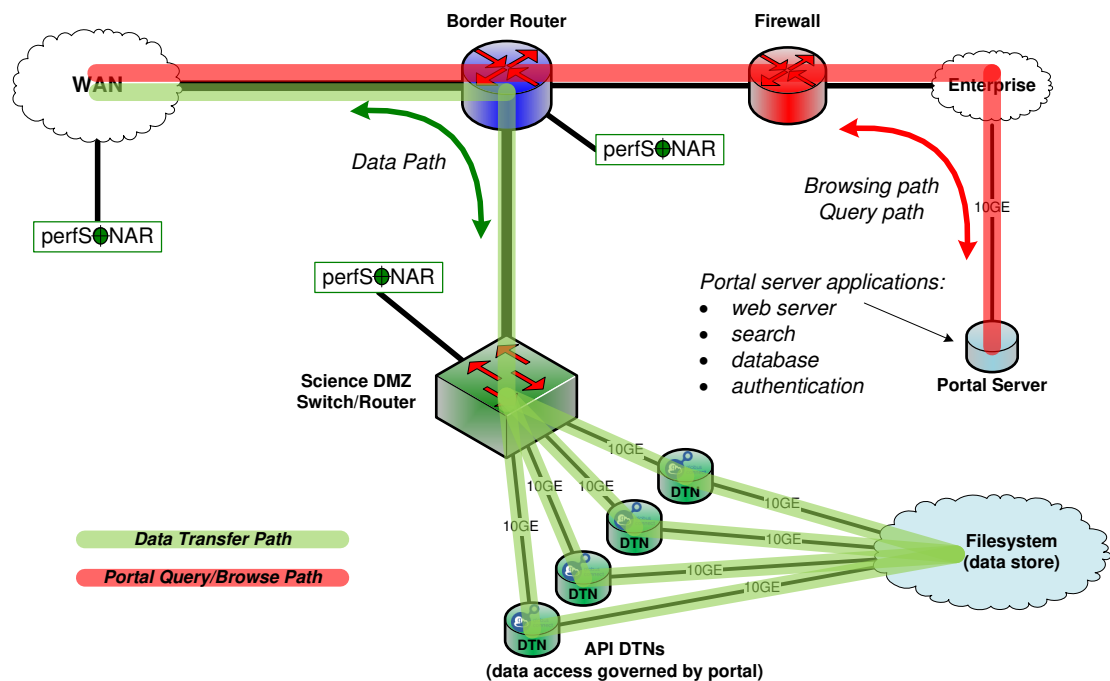


Figure 2. The MRDP design pattern from a network architecture perspective. The Science DMZ includes multiple DTNs that provide for high-speed transfer between network and storage. Portal functions run on a portal server, located on the institution's enterprise network. The DTNs need only speak the API of the data management service (Globus in this case).

85 over the past decade, at rates far faster than can be supported by monolithic web services applications
 86 running on individual servers connected to enterprise networks and secured by enterprise firewalls.
 87 In response, network architectures are proliferating to allow high-speed access to research data over
 88 separate data channels that are specifically engineered for high-performance data services. However, these
 89 developments do not mesh well with the LRDP architecture. The LRDP server typically cannot be moved
 90 outside the institutional firewall because it is a complex web services application with sensitive information
 91 (e.g., user databases) that must be defended using correspondingly complex security protections. Also,
 92 the LRDP model forces all data objects through the web server software stack, limiting performance and
 93 scalability.

94 The second is an increasing demand for **high data transfer reliability**. When download or upload
 95 requests involve just a few small files, transient errors are rare and can be handled by having users resubmit
 96 requests. When requests involve thousands or more files and gigabytes or terabytes, errors are more
 97 problematic. Increasingly, researchers expect the high reliability offered by transfer services such as
 98 Globus (Chard et al., 2014), which use specialized techniques to minimize the impact of errors. But
 99 retrofitting such mechanisms into the LRDP model is challenging.

100 The third challenge is **operational complexity**, which in turn can negatively impact development
 101 costs, reliability, capabilities offered, and security. Conventional LRDP implementations are developed,
 102 deployed, and operated in silos, with each establishing and operating its own implementations of user
 103 management, authentication, authorization, and data transfer. This siloed approach is sub-optimal
 104 and inefficient, and often results in limited functionality relative to state-of-the-art. For example, few
 105 legacy portals enable authentication and authorization with standard web protocols such as OpenID
 106 Connect (Sakimura et al., 2014) and OAuth 2 (Hardt, 2012), instead preferring to manage local user and
 107 authorization databases. The siloed approach makes it difficult to ensure that best practices, especially
 108 with respect to security, are followed. It also increases the burden on administrators to ensure that not
 109 only is the portal available but also that updates and security patches are applied quickly.

110 In summary, while the LRDP model has served the scientific community well for many years, it
 111 suffers from fundamental limitations, with the result that portals based on the LRDP model serve the

112 scientific community less well than they otherwise could.

113 THE MRDP DESIGN PATTERN

114 This discussion of LRDP limitations brings us to the core of this article, namely, the modern research
115 data portal (MRDP) design pattern. This new approach to building research data portals became feasible
116 in around 2015, following the maturation of two independent efforts aimed at decoupling research data
117 access from enterprise networks, on the one hand, and research data management logic from storage
118 system access, on the other. These developments inspired a new approach to the construction of research
119 data portals based on a decomposition of the previously monolithic LRDP architecture into three distinct
120 components:

- 121 1. The **portal server** (a web server like any other) which handles data search and access, mapping
122 between users and datasets, and other web services tasks;
- 123 2. A high-performance network enclave that connects large-scale data servers directly to high-
124 performance networks (we use the **Science DMZ** as an example here); and
- 125 3. a reliable, high-performance external data management service with authentication and other
126 primitives based on standard web APIs (we use **Globus** as an example here).

127 In the remainder of this section, we describe the role of the Science DMZ and the data servers that
128 reside within it, the role of Globus as a provider of outsourced data management and security services,
129 and the integration of these components to form the MRDP design pattern.

130 Science DMZ and DTNs

131 A growing number of research institutions are connected to high-speed wide area networks at high speeds:
132 10 gigabits per second (Gb/s) or faster. Increasingly, these wide area networks are themselves connected
133 to cloud providers at comparable speeds. Thus, in principle, it should be possible to move data between
134 any element of the national research infrastructure—between institutions, laboratories, instruments, data
135 centers, supercomputer centers, and clouds—with great rapidity.

136 In practice, real transfers often achieve nothing like these peak speeds. Common reasons for poor
137 performance are the complexity and architectural limitations of institutional networks, as well as complex
138 and inefficient configurations on monolithic server computers. Commodity network devices, firewalls,
139 and other limitations cause performance bottlenecks in the network between the data service and the
140 outside world where clients are located.

141 Two constructs, the Science DMZ and the Data Transfer Node, are now widely deployed to overcome
142 this problem. As shown in Figure 2, the **Science DMZ** overcomes the challenges associated with multi-
143 purpose enterprise network architectures by placing resources that need high-performance connectivity
144 in a special subnetwork that is close (from a network architecture perspective) to the border router that
145 connects the institution to the high-speed wide area network. (The term DMZ, short for demilitarized
146 zone, is commonly used in computer networking to indicate an intermediate role between external and
147 internal networks.) Traffic between those resources and the outside world then has a clean path to the
148 analogous high-performance resources at collaborating institutions.

149 A **Data Transfer Node** (DTN) is a specialized device dedicated to data transfer functions. These
150 devices are typically PC-based Linux servers constructed with high quality components, configured for
151 both high-speed wide area data transfer and high-speed access to local storage resources, and running
152 high-performance data transfer tools such as Globus Connect data transfer software. General-purpose
153 computing and business productivity applications, such as email clients and document editors, are not
154 installed; this restriction produces more consistent data transfer behavior and makes security policies
155 easier to enforce.

156 The Science DMZ design pattern also includes other elements, such as integrated monitoring devices
157 for performance debugging, specialized security configurations, and variants used to integrate super-
158 computers and other resources. For example, Figure 2 shows perfSONAR (Hanemann et al., 2005)
159 performance monitoring devices. But this brief description provides the essential information required for
160 our discussion here. The US Department of Energy's Energy Sciences Network has produced detailed
161 configuration and tuning guides for Science DMZs and DTNs (ESnet, 2017).

162 **Globus services**

163 Globus provides data and identity management capabilities designed for the research community. These
164 capabilities are delivered via a cloud-hosted software- and platform-as-a-service model, enabling users to
165 access them through their web browser and developers to invoke them via powerful APIs. We describe
166 here Globus capabilities that meet MRDP needs for managing and transferring data (Chard et al., 2014)
167 and for authenticating users and authorizing access (Tuecke et al., 2015).

168 Globus allows data to be remotely managed across its pool of more than 10,000 accessible storage
169 systems (called “endpoints”). A storage system is made accessible to Globus, and thus capable of high
170 performance and reliable data transfer, by installing Globus Connect software. Globus Connect is offered
171 in two versions: Globus Connect Personal for single-user deployments (e.g., a laptop or PC) and Globus
172 Connect Server for multi-user deployments (e.g., a shared server or DTN). Globus Connect Server can
173 be deployed on multiple DTNs associated with a storage system; Globus then uses the pool of DTNs to
174 increase transfer performance, with dynamic failover for increased reliability.

175 **Globus Transfer** capabilities provide high performance and reliable third party data transfer. The
176 Globus service manages the entire transfer process, including coordinating authentication at source and
177 destination; establishing a high performance data channel using the GridFTP (Allcock et al., 2005) proto-
178 col, with configuration optimized for transfer; ensuring data integrity by comparing source/destination
179 checksums; and recovering from any errors during the transfer. Globus also provides secure (authorized)
180 HTTPS access to (upload/download) data via a web browser or an HTTP command line client (e.g., for
181 small files, inline viewers, or transitional support of LRDP usage models). Globus, by default, enforces
182 the data access permissions represented by the underlying system; however, it also allows these access
183 decisions to be managed through the cloud service. In the latter mode, called **Globus Sharing** (Chard
184 et al., 2014), users may associate user- or group-based access control lists (ACLs) with particular file
185 paths. Globus checks and enforces these ACLs when other users attempt to read or write to those paths.

186 **Globus Auth** provides identity and access management platform capabilities. It brokers authentication
187 and authorization interactions between end-users, identity providers, resource servers (services), and
188 clients (e.g., web, mobile, desktop, and command line applications, and other services). It implements
189 standard web protocols, such as OAuth 2 and OpenID Connect, that allow it to be integrated easily with
190 external applications using standard client libraries. These protocols enable third-party applications to
191 authenticate users (using their preferred identity) directly with the chosen identity provider. Globus Auth
192 then returns access tokens that the third-party application can use to validate the user’s identity and to
193 perform actions on behalf of that user, within an agreed upon scope. Globus Auth implements an identity
194 federation model via which diverse identities can be linked, and such that presentation of one identity
195 may support authorization for the set of identities. Integration with Globus Groups (Chard et al., 2016)
196 supports group-based authorization using user-managed groups.

197 REST APIs allow Globus capabilities to be used as a platform. These APIs are used by client libraries,
198 such as the Globus Python SDK, to support integration in external applications. We leverage the Globus
199 Python SDK in the MRDP reference implementation that we describe in this paper, and in the code
200 examples presented in subsequent sections.

201 **The design pattern in practice**

202 The MRDP design pattern improves on the LRDP architecture in three important ways. While these
203 improvements involve relatively minor changes to the web server logic, they have big implications for
204 how data accesses and transfers are performed.

205 First, **data references**. An important interaction pattern in a research data portal is the redirect, in
206 which a client request for information leads to the return of one or more URLs for data files. In the LRDP
207 pattern, these are web URLs that reference files served by the same web server that runs the portal GUI.
208 In the MRDP pattern, the portal instead returns references to data objects served by DTNs located in a
209 Science DMZ separate from the portal web server, as shown in Figure 2. This approach allows the data
210 objects to be transferred using infrastructure that matches their scale. Also, because the DTN cluster can
211 be expanded (e.g., by adding more or faster DTNs) without changing the interface that it provides to the
212 portal, it can easily be scaled up as datasets and traffic volumes grow, without modifying the portal code
213 at all.

214 Second, **data access**. In the LRDP pattern, the references to data objects are returned to the user in a
215 web page. The user must then access the data objects by clicking on links or by using a web command

216 line client like `wget`. Neither approach is convenient or reliable when dealing with many files. In the
217 MRDP pattern, references are encapsulated in a data transfer job which can be managed by a cloud-based
218 data transfer service such as Globus. Thus the user can hand off the complexity of managing the correct
219 transfer of perhaps many thousands of files to a data management service that is designed to handle such
220 tasks reliably.

221 Third, **user and group management**, and **authentication and authorization**. In the LRDP pattern,
222 these functions are typically all hosted on the web server. Portal developers must therefore implement
223 workflows for authenticating users, requesting access to data, assembling datasets for download, dy-
224 namically authorizing access to data, and checking data integrity (e.g., by providing checksums). In the
225 MRDP pattern, these functions can be outsourced via standard interfaces to external services that provide
226 best-practices implementations.

227 An important feature of the MRDP design pattern is the use of web service (REST) APIs when
228 accessing external services that provide data management and other capabilities. This approach makes
229 it straightforward to retrofit advanced MRDP features into an existing data portal without requiring a
230 rewrite of the data portal software from scratch.

231 Variants of the basic pattern

232 In the following, we present a reference implementation of the MRDP design pattern that enables
233 download of data selected on a web page. Many variants of this basic MRDP design pattern have been
234 constructed. For example, the data that users access may come from an experimental facility rather than
235 a data archive, in which case they may be deleted after download. Access may be granted to groups of
236 users rather than individuals. Data may be publicly available; alternatively, access may require approval
237 from portal administrators. A portal may allow its users to upload datasets for analysis and then retrieve
238 analysis results. A data publication portal may accept data submissions from users, and load data that
239 pass quality control procedures or curator approval into a public archive. We give examples of several
240 such variants below, and show that each can naturally be expressed in terms of the MRDP design pattern.

241 Similarly, while we have described the research data portal in the context of a Science DMZ, in
242 which (as shown in Figure 2) the portal server and data store both sit within a research institution, other
243 distributions are also possible and can have advantages. For example, the portal can be deployed on the
244 public cloud for high availability, while data sit within a research institution's Science DMZ to enable
245 direct access from high-speed research networks and/or to avoid cloud storage charges. Alternatively, the
246 portal can be run within the research institution and data served from cloud storage. Or both components
247 can be run on cloud resources.

248 A REFERENCE MRDP IMPLEMENTATION

249 We have developed a reference implementation of the MRDP design pattern, for which open source code
250 is available on the companion web site. This code includes:

- 251 • A complete, working portal server, implemented with the Python Flask framework and comprising a
252 web service and web interface, and that uses Globus APIs to outsource data transfer, authentication,
253 and authorization functions.
- 254 • Integration with Globus Auth for authentication and authorization.
- 255 • Integration with Globus Transfer for browsing and downloading datasets.
- 256 • Use of a decoupled Globus endpoint for serving data securely via HTTP or GridFTP.
- 257 • An independent analysis service, accessed via a REST API, to demonstrate how a data portal can
258 outsource specific functionality securely.

259 We review some highlights of this reference implementation here; more details are available on the
260 companion web site.

261 Figure 3 shows the essential elements of this reference implementation. In addition to the Science
262 DMZ and portal server already seen in Figure 2, we see the user's desktop computer or other device
263 on which they run the web browser (or other client) used to access the data portal, plus other elements
264 (identity providers, Globus cloud, other services) that we introduce in the following. Note the "Globus

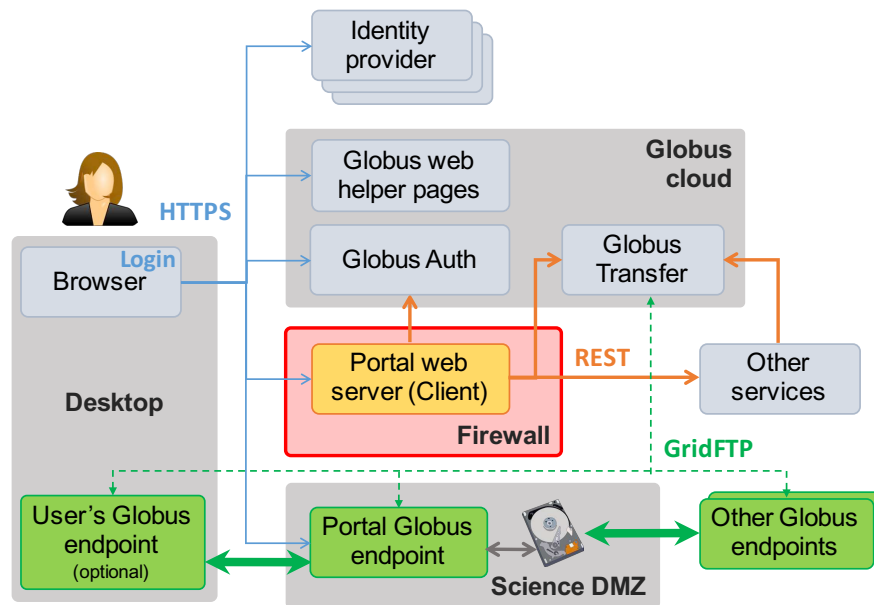


Figure 3. MRDP basics. Clients (left) authenticate with any one of many identity providers (top) and connect to the portal web server (center) that implements the domain-specific portal logic. The web server sits behind the institutional firewall (red). The portal responds to client requests by using REST APIs to direct Globus cloud services (top) to operate on research data in the Science DMZ (bottom center) and/or to interact with other services (center right). Data flows reliably and securely between Globus endpoints without traversing the firewall.

265 endpoints” (Globus Connect servers) located in the Science DMZ, on other remote storage systems, and
 266 (optionally) on the user’s desktop. The cloud-hosted Globus transfer service orchestrates high-speed,
 267 reliable GridFTP transfers among these endpoints, under the direction of the portal server.

268 As already noted, the **portal server** is at the heart of the MRDP implementation. It sits behind the
 269 institutional firewall, from where it serves up web pages to users, responds to HTTP requests, and issues
 270 REST communications to Globus services and optionally other services to implement MRDP behaviors.
 271 The latter REST communications are central to the power of the MRDP design pattern, as it is they that
 272 let the web server outsource many of the complex tasks associated with portal operations. Only the web
 273 server component needs to be provided for a specific MRDP implementation.

274 Overview of key points

275 The MRDP design pattern employs a collection of modern approaches to delivering critical capabilities.
 276 We review some important points here, with comments on how they can be realized with Globus
 277 mechanisms. More specifics are on the companion web site.

278 *Outsource responsibility for determining user identities.* Operating an identity provider should not
 279 need to be a core competency for a portal administrator. Globus Auth support for the OAuth 2 and
 280 OpenID protocols allows for the validation of credentials from any one of a number of identity providers
 281 deemed acceptable by the portal: for example, institutional credentials via InCommon (Barnett et al.,
 282 2011), ORCID, or Google identities.

283 *Outsource control over who can access different data and services within the portal.* Nor should the
 284 portal developer or administrator need to be concerned with implementing access control mechanisms.
 285 The Globus transfer service can be used to control who is allowed to access data.

286 *Outsource responsibility for managing data uploads and downloads between a variety of locations
 287 and storage systems.* Reliable, efficient, and secure data transfer between endpoints is a challenging task
 288 to do well. Again, simple APIs allow this task to be handed off to the Globus transfer service. (A portal
 289 can also leverage Globus HTTPS support to provide web-based download and inline viewers.) Thus, the
 290 portal does not need to provide its own (typically unreliable and/or slow) data download client, as some

291 portals do.

292 *Leverage standard web user interfaces for common user actions.* The implementation of the portal's
293 web interface can be simplified via the use of standard components. Globus provides web helper pages
294 for such tasks as selecting endpoints, folders, and files for transfers; managing group membership; and
295 logging out.

296 *Dispatch tasks to other services on behalf of requesting users.* Good programming practice often
297 involves decomposing a complex portal into multiple services. Globus Auth *dependent grant flows* (an
298 OAuth concept) enable a portal to allow other services to operate on managed data. We discuss below how
299 the reference implementation uses this mechanism to allow an external data analysis service to access a
300 user's data.

301 *Log all actions performed by users for purposes of audit, accounting, and reporting.* The portal should
302 stores a historical log of all actions performed by the portal and its users such that others can determine
303 what data has been accessed, when, and by whom. Again, these functions can be outsourced to a cloud
304 service.

305 **Diving into code**

306 The reference implementation code is for a portal that allows users to browse for data of interest and then
307 request those data for download. The portal makes the data available via four simple steps: (1) create a
308 shared endpoint; (2) copy the requested data to that shared endpoint; (3) set permissions on the shared
309 endpoint to enable access by the requesting user, and email the user a URL that they can use to retrieve
310 data from the shared endpoint; and ultimately (perhaps after days or weeks), (4) delete the new shared
311 endpoint.

312 Listing 1 presents a function `rdp` that uses the Globus Python SDK to implement these actions. This
313 code is somewhat simplified: it does not perform error checking and does not integrate with the web
314 server that our data portal also requires, but otherwise it is a complete implementation of the core portal
315 logic—in just 42 lines of code. In the following, we first review some Globus background and then
316 describe some elements of this program.

317 **Endpoints**

318 Figure 3 shows several *Globus endpoints*. These are Globus Connect servers deployed next to storage
319 systems to enable high-speed, reliable, and secure access to those storage systems. They may be deployed
320 within a Science DMZ, on a user's personal computer, on a cloud system, or elsewhere. Each endpoint
321 implements the GridFTP protocol (thick green arrows) for high-speed endpoint-to-endpoint data transfer
322 and the HTTPS protocol (blue arrows) to enable access to endpoint storage from a web client. Endpoints
323 are managed by the Globus Transfer cloud service via the GridFTP control channel: the thin green dashed
324 lines.

325 In order for Globus Transfer to perform operations on an endpoint filesystem, it must have a credential
326 to authenticate to the endpoint as a specific local user. The process of providing such a credential to
327 the service is called *endpoint activation*. Endpoints typically require credentials from one or more
328 specific identity providers (IdPs). A Globus Connect Server can be configured with a co-located MyProxy
329 OAuth (Basney and Gaynor, 2011) server to allow local user accounts to be used for authentication.
330 Alternatively, endpoints may be configured to use one of the supported Globus Auth identity providers. In
331 this case, endpoints contain a mapping between identity provider identities and local user accounts. Each
332 Globus endpoint (and each user) is named by a universally unique identifier (UUID).

333 **Identities and credentials**

334 We see from Figure 3 that the MRDP design pattern involves interactions among numerous entities: user,
335 web client, Globus services, portal server, Globus endpoints, and perhaps other services as well. Each
336 interaction requires authentication to determine the originator of the request and access management to
337 determine whether the request should be granted.

338 The first challenge is the need to allow users to authenticate with the portal. To reduce identity prolif-
339 eration, it is desirable that the portal allow for authentication using existing accounts (e.g., an institution
340 account). This is also advantageous to the portal as it need not implement an identity management system,
341 keep up with best-practices security models, store user passwords, and facilitate identity management
342 workflows (e.g., password reset). While supporting existing identities is indeed desirable, the task is

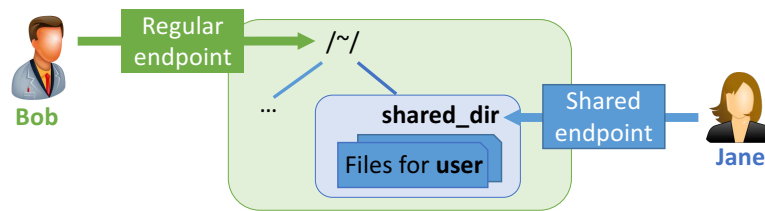


Figure 4. A shared endpoint scenario.

343 complicated by the ever growing set of providers, and the wide variety of protocols designed to allow
 344 users to verify ownership of an identity.

345 Access management decisions are complicated by the fact that a single user may possess, and
 346 need to use, multiple identities from different IDPs. For example, a user Jane Doe with an identity
 347 `jane@uni.edu` at her university and another identity `jdove@lab.gov` at a national laboratory may need
 348 to use both identities when transferring a file from endpoint A at `uni.edu` to endpoint B at `lab.gov`.

349 The Globus Auth identity and access management service manages these complexities. It can be used
 350 as an identity broker, allowing users to authenticate using one of hundreds of supported identity providers
 351 (IDPs) and enabling integration in third-party services via a single standard protocol (e.g., OAuth 2). It
 352 also allows users to link their different identities. When Jane uses a web client to log in to Globus, Globus
 353 Auth requests that she verify her identity via one of her linked identity providers (e.g., by obtaining and
 354 checking a password). This verification is performed via a redirection workflow (e.g., OAuth 2) to ensure
 355 that the password used for verification is never seen by Globus Auth. Upon successful authentication,
 356 Globus Auth creates a set of unique, fixed-term access tokens and returns them to the requesting client.
 357 Each access token specifies the time period and purpose for which it can be used (e.g., transferring
 358 data, accessing profile information). The web client can then associate a token with subsequent HTTP
 359 requests—for example a transfer request to Globus Transfer—to demonstrate that Jane has verified her
 360 identity.

361 Let's say that Jane now uses the Globus Transfer web client to request a file transfer from endpoint
 362 A at `uni.edu` to endpoint B at `lab.gov`. Endpoint A requires a `uni.edu` access token, which the web
 363 client can provide due to the initial authentication. Endpoint B requires a `lab.gov` access token, which
 364 the web client does not possess. The Globus Transfer web client will then ask Globus Auth to invoke a
 365 further OAuth 2 operation to verify that identity and obtain a second access token.

366 The MRDP implementation employs a Globus capability called the *shared endpoint*, a dynamically
 367 created construct that enables a folder on an existing endpoint to be shared with other Globus users. To
 368 use this feature, a user authorized to operate on the existing endpoint first creates a shared endpoint,
 369 designating the existing endpoint and folder; then grants read and/or write permissions on the shared
 370 endpoint to the Globus user(s) and/or group(s) that they want to access it. Those users can then access
 371 that endpoint like any other endpoint, but do not need a local account.

372 This construct is illustrated in Figure 4. *Bob* has enabled sharing of folder `~/shared_dir` on Regular
 373 endpoint to create Shared endpoint, and then granted *Jane* access to that shared endpoint. *Jane* can
 374 then use Globus to read and/or write files in the shared folder, depending on what rights she has been
 375 granted.

376 The `rdp` function

377 The three arguments to the function `rdp` in Listing 1 can be understood in terms of Figure 4. Those
 378 arguments are, in turn, the UUID for the endpoint on which the shared endpoint is to be created, the name
 379 of the folder on that endpoint from which the contents of the shared folder are to be copied, and the email
 380 address for the user who is to be granted access to the shared endpoint. In this case we use the Globus
 381 endpoint named "Globus Tutorial Endpoint 1" and sample data available on that endpoint.

```
rdp('ddb59aef-6d04-11e5-ba46-22000b92c6ec',
    '~/share/godata/',
    'jane@uni.edu')
```

382 The code in Listing 1 uses the Globus Python SDK to create, manage access to, and delete the shared
383 endpoint, as follows. It first creates a `TransferClient` and an `AuthClient` object—classes provided
384 by the Globus Python SDK for accessing the Globus Transfer and Globus Auth services, respectively.
385 Each class provides a rich set of methods for accessing the various resources defined in the REST APIs.
386 We then use the SDK function `endpoint_autoactivate` to ensure that the portal has a credential that
387 permits access to the endpoint identified by `host_id`.

388 In Step 1(a), we use the Globus SDK function `operation_mkdir` to create a directory (named, in our
389 example call, by a UUID) on the endpoint with identifier `host_id`. Then, in Step 1(b), the SDK function
390 `create_shared_endpoint` is used to create a shared endpoint for the new directory. At this point, the
391 new shared endpoint exists and is associated with the new directory, but only the creating user has access.

392 In Step 3, we first use the Globus SDK function `get_identities` to retrieve the user identifier
393 associated with the supplied email address; this is the user for whom sharing is to be enabled. (If this user
394 is not known to Globus, an identity is created.) We then use the function `add_endpoint_acl_rule` to
395 add to the new shared endpoint an access control rule that grants the specified user read-only access to the
396 endpoint.

397 As our `add_endpoint_acl_rule` request specifies an email address, an invitation email is sent to the
398 user. At this point, the user is authorized to download data from the new shared endpoint. The shared
399 endpoint is typically left operational for some period and then deleted, as shown in Step 4. Note that
400 deleting a shared endpoint does not delete the data that it contains. The portal admin may want to retain
401 the data for other purposes. If not, we can use the Globus SDK function `submit_delete` to delete the
402 folder.

403 **Data transfer**

404 We skipped over Step 2 of Listing 1 in the preceding discussion. That step requests Globus to transfer the
405 contents of the folder `source_path` to the new shared endpoint. (The transfer in the code is from the
406 endpoint on which the new shared endpoint has been created, but it could be from any Globus endpoint
407 that the portal administrator is authorized to access.) The code assembles and submits the transfer request,
408 providing the endpoint identifiers, source and destination paths, and (as we want to transfer a directory)
409 the recursive flag. It then waits for either the transfer to complete or for a specified timeout to elapse. In
410 practice, the code should check for the response code from the `task_wait` call and then repeat the wait
411 or terminate the transfer on error or timeout.

412 **Web and command line interfaces**

413 Having received an email invitation to access the new shared endpoint, the user can click on the embedded
414 URL to access that endpoint via the Globus web interface. Users can then transfer the data from that
415 endpoint to any other endpoint to which they are authorized to transfer.

416 The Globus web interface also provides access to all other Globus functions, allowing users to create
417 shared endpoints, manage access control to those endpoints, and delete a shared endpoint when it is
418 no longer needed. Users may also use the Python SDK or REST APIs to perform these same actions
419 programmatically. In addition, a Globus command line interface (CLI), implemented via the Python SDK,
420 can be used to perform the operations just described.

421 **Completing the MRDP portal server**

422 Our MRDP reference implementation consists of a simple research data portal server that allows users
423 to sign up, log in, select datasets from a set of temperature record datasets, and then either download
424 or request the creation of graphs for the selected datasets. This complete implementation is some 700
425 lines of code, plus 500 lines for an example service that we discuss in the next section. A screenshot of a
426 deployed version of this code is shown in Figure 5.

427 The portal server implementation uses the Python Flask web framework, a system that makes it
428 easy to create simple web servers. The core Flask code defines, via @-prefixed *decorators*, what actions
429 should be performed when a user accesses specific URLs. These *views* support both authentication and
430 the core MRDP logic as well as additional functionality for managing profiles. Several views generate
431 template-based web pages that are customized for user interactions.

432 The OAuth 2 authentication workflow proceeds as follows. When logging in, (`login()`) the user is
433 redirected to Globus Auth to authenticate with their identity provider. The resulting access code is returned
434 to the portal, where it is exchanged for access tokens that can be used to interact with Globus Auth or

Listing 1. Globus code to implement MRDP design pattern

```
from globus_sdk import TransferClient, TransferData
from globus_sdk import AuthClient
import sys, random, uuid

def rdp(host_id,      # Endpoint for shared endpoint
        source_path, # Directory to copy data from
        email):     # Email address to share with
    tc = TransferClient()
    ac = AuthClient()
    tc.endpoint_autoactivate(host_id)

    # (1) Create shared endpoint:
    # (a) Create directory to be shared
    share_path = '/~/ ' + str(uuid.uuid4()) + '/'
    tc.operation_mkdir(host_id, path=share_path)
    # (b) Create shared endpoint on directory
    shared_ep_data = {
        'DATA_TYPE': 'shared_endpoint',
        'host_endpoint': host_id,
        'host_path': share_path,
        'display_name': 'RDP shared endpoint',
        'description': 'RDP shared endpoint'
    }
    r = tc.create_shared_endpoint(shared_ep_data)
    share_id = r['id']

    # (2) Copy data into the shared endpoint
    tc.endpoint_autoactivate(share_id)
    tdata = TransferData(tc, host_id, share_id,
                        label='RDP copy', sync_level='checksum')
    tdata.add_item(source_path, '/', recursive=True)
    r = tc.submit_transfer(tdata)
    tc.task_wait(r['task_id'], timeout=1000,
                polling_interval=10)

    # (3) Enable access by user
    r = ac.get_identities(usernames=email)
    user_id = r['identities'][0]['id']
    rule_data = {
        'DATA_TYPE': 'access',
        'principal_type': 'identity', # Grantee is
        'principal': user_id,        # a user.
        'path': '/',                 # Path is /
        'permissions': 'r',          # Read-only
        'notify_email': email,       # Email invite
        'notify_message':            # Invite msg
            'Requested data are available.'
    }
    tc.add_endpoint_acl_rule(share_id, rule_data)

    # (4) Ultimately, delete the shared endpoint
    tc.delete_endpoint(share_id)
```

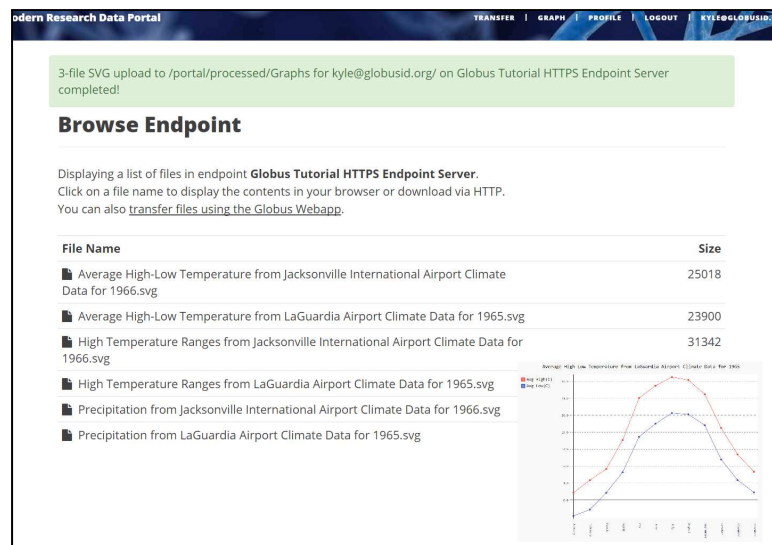


Figure 5. Example MRDP showing a list of computed graphs for a user and (inset) one of these graphs.

435 other dependent services (e.g., Globus Transfer). Access tokens are then validated and introspected to
436 obtain user information.

437 The portal then provides two methods for accessing data. The first method allows users to download
438 raw datasets from a list of available datasets. In this case, it makes a call to Globus Transfer to retrieve a
439 real-time listing of directories in its associated endpoint. The second method allows users to dynamically
440 create graphs based on selected datasets (by year and id), as described in the following subsection. The
441 resulting graph is stored in the portal's shared endpoint in a directory accessible only to the requesting user.
442 Irrespective of which method is used, users may download data (i.e., raw datasets or computed graphs)
443 via HTTP (using a direct URL to the object on the portal's endpoint) or by transferring them to another
444 endpoint. In the later case, the user is prompted for a destination endpoint. Rather than re-implement
445 endpoint browsing web pages already offered by Globus, the portal instead uses Globus "helper pages" to
446 allow users to browse the endpoint. The helper page returns the endpoint ID and path to the portal. The
447 portal then starts the transfer using the Globus Transfer API.

448 The following code, from the portal implementation file `portal/view.py`, specifies that when a user
449 visits `http://localhost:5000/login`, the user should be redirected to the `authcallback` URL
450 suffix.

```
451 @app.route('/login', methods=['GET'])
452 def login():
453     """Send the user to Globus Auth."""
454     return redirect(url_for('authcallback'))
```

455 The URL `http://localhost:5000/authcallback` in turn calls the `authcallback` function,
456 shown in Listing 2, which uses the OAuth 2 protocol to obtain access tokens that the portal can sub-
457 sequently use to interact with Globus Auth or dependent services (e.g., Globus Transfer or the graph
458 service.) The basic idea is as follows. First, the web server redirects the user to authenticate using Globus
459 Auth. The redirect URL includes the URL to return to (`http://localhost:5000/authcallback`)
460 after the user has authenticated. The response includes a `auth code` parameter which can be unpacked
461 and then swapped for access tokens by contacting Globus Auth and specifying the scopes needed by the
462 portal. Finally, the resulting access tokens are returned to the portal in a JSON object which also includes
463 information about the user's identity.

464 The last line returns, redirecting the web browser to the portal's transfer page, as shown in Figure 6.

465 A request to transfer files requires that the user first select the dataset(s) to be transferred and then
466 specify the destination endpoint and location for the dataset(s). Listing 3 implements these behaviors.
467 First, the code checks that the user has selected datasets on the transfer web page. Then, the code redirects
468 the user to `https://www.globus.org/app/browse-endpoint`, one of the *web helper pages* that
469 Globus operates to simplify MRDP implementation. The browse endpoint helper page returns the endpoint

Listing 2. The `authcallback` function interacts with Globus Auth to obtain access tokens for the server.

```
@app.route('/authcallback', methods=['GET'])
def authcallback():
    # Handles the interaction with Globus Auth
    # Set up our Globus Auth/OAuth 2 state
    redirect_uri = url_for('authcallback', _external=True)

    client = load_portal_client()
    client.oauth2_start_flow_authorization_code(redirect_uri, refresh_tokens=True)

    # If no "code" parameter, we are starting
    # the Globus Auth login flow
    if 'code' not in request.args:
        auth_uri = client.oauth2_get_authorize_url()
        return redirect(auth_uri)
    else:
        # If we have a "code" param, we're coming
        # back from Globus Auth and can exchange
        # the auth code for access tokens.
        code = request.args.get('code')
        tokens = client.oauth2_exchange_code_for_tokens(code)

        id_token = tokens.decode_id_token(client)
        session.update(
            tokens=tokens.by_resource_server,
            is_authenticated=True,
            name=id_token.get('name', ''),
            email=id_token.get('email', ''),
            project=id_token.get('project', ''),
            primary_username=id_token.get('preferred_username'),
            primary_identity=id_token.get('sub'),
        )

    return redirect(url_for('transfer'))
```

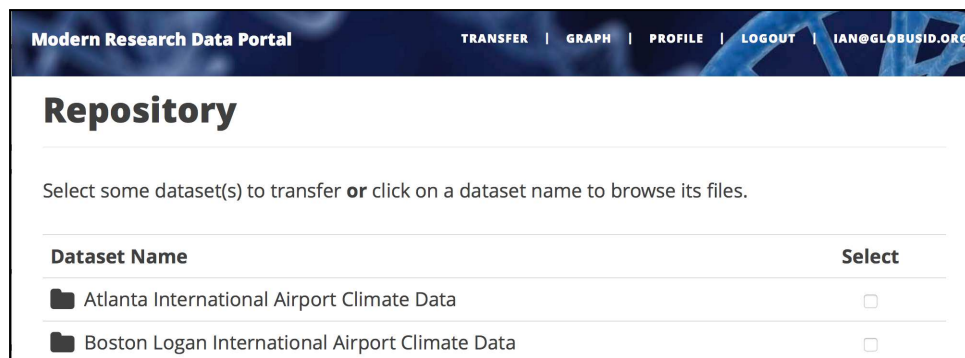


Figure 6. A portion of the MRDP reference implementation, showing the five user options at top (each mapped to a “route” in the code) and two of the available datasets.

470 ID and path to which the user wants to transfer the selected dataset(s). The `submit_transfer` function
 471 (not shown here) uses the endpoint ID and path to execute a Globus transfer request using code similar to
 472 Step 2 in Listing 1.

473 Invoking other services

474 The final element of the MRDP design pattern that we discuss here is the invocation of other services, as
 475 shown by the arrow labeled REST from the portal server to **Other services** in Figure 3. Such calls might
 476 be used in an MRDP instance for several reasons. You might want to organize your portal as a lightweight
 477 front end (e.g., pure Javascript) that interacts with one or more remote backend (micro)services. You
 478 might want to provide services that perform subsetting, quality control, data cleansing, or other analyses
 479 before serving data. Another reason is that you might want to provide a public REST API for the main
 480 portal machinery, so that other app and service developers can integrate with and build on your portal.

481 Our reference implementation illustrates this capability. The data portal skeleton allows a client to
 482 request that datasets be graphed (`graph()`). It does not perform those graphing operations itself but
 483 instead sends a request to a separate Graph service. The request provides the names of the datasets to be
 484 graphed. The Graph service retrieves these datasets from a specified location, runs the graphing program,
 485 and uploads the resulting graphs to a dynamically created shared endpoint for subsequent retrieval. The
 486 reference implementation includes a complete implementation of the Graph service, showing how it
 487 manages authentication and data transfer with Globus APIs.

488 EXAMPLES OF THE MRDP DESIGN PATTERN

489 We briefly present five examples of large-scale implementations of the MRDP design pattern. We present
 490 performance results for several of these examples in the next section.

491 The NCAR Research Data Archive

492 The Research Data Archive (RDA) operated by the US National Center for Atmospheric Research at
 493 <http://rda.ucar.edu> contains more than 600 data collections, ranging in size from megabytes to
 494 tens of terabytes. These collections include meteorological and oceanographic observations, operational
 495 and reanalysis model outputs, and remote sensing datasets to support atmospheric and geosciences
 496 research, along with ancillary datasets, such as topography/bathymetry, vegetation, and land use. The
 497 RDA data portal allows users to browse and search data catalogs, and then download selected datasets to
 498 their personal computer or HPC facility.

499 RDA users are primarily researchers at federal and academic research laboratories. In 2016 alone,
 500 more than 24,000 people downloaded more than 1.9 petabytes. The RDA portal thus requires robust,
 501 scalable, maintainable, and performant implementations of a range of functions, some domain-independent
 502 (e.g., user identities, authentication, and data transfer) and others more domain-specific (e.g., a catalog of
 503 environmental data collections).

504 RDA uses the techniques described previously to implement the MRDP design pattern, except that
 505 they do not currently outsource identity management. The use of the MRDP design pattern for RDA

Listing 3. The `transfer()` function from the web server reference implementation.

```

@app.route('/transfer', methods=['GET', 'POST'])
@authenticated
def transfer():
    if request.method == 'GET':
        return render_template('transfer.jinja2', datasets=datasets)

    if request.method == 'POST':
        # Check that file(s) have been selected for transfer
        if not request.form.get('dataset'):
            flash('Please select at least one dataset.')
            return redirect(url_for('transfer'))

        params = {
            'method': 'POST',
            'action': url_for('submit_transfer', _external=True, _scheme='https'),
            'filelimit': 0,
            'folderlimit': 1
        }

        browse_endpoint =
            'https://www.globus.org/app/browse-endpoint?{}'.format(urlencode(params))

        # Save submitted form to session
        session['form'] = {
            'datasets': request.form.getlist('dataset')
        }

        # Send to Globus to select a destination endpoint using
        # the Browse Endpoint helper page.
        return redirect(browse_endpoint)

```

506 allows for vastly increased scalability in terms of dataset size, and much lower human effort for managing
 507 the transfer of multi-terabyte datasets to computing centers for analysis.

508 Sanger Imputation Service

509 This service, operated by the Wellcome Trust Sanger Institute at <https://imputation.sanger.ac.uk>, allows you to upload files containing genome wide association study (GWAS) data from the
 510 23andMe genotyping service and receive back the results of imputation and other analyses that identify
 511 genes that you are likely to possess based on those GWAS data (McCarthy et al., 2016). This service uses
 512 Globus APIs to implement a variant of the MRDP design pattern, as follows.

514 A user who wants to use the service first registers an imputation job. As part of this process, they are
 515 prompted for their name, email address, and identity, and the type of analysis to be performed. The portal
 516 then requests Globus to create a shared endpoint, share that endpoint with the identity provided by the
 517 user, and email a link to this endpoint to the user. The user clicks on that link to upload their GWAS data
 518 file and the corresponding imputation task is added to the imputation queue at the Sanger Institute. Once
 519 the imputation task is completed, the portal requests Globus to create a second shared endpoint to contain
 520 the output and to email the user a link to that new endpoint for download. The overall process differs
 521 from that of Listing 1 only in that a shared endpoint is used for data upload as well as download.

522 User-managed data sharing portals

523 Argonne National Laboratory's Petrel (<http://petrel.alcf.anl.gov/>) implements a special-
 524 ized research data portal that allows users to request a space allocation and then upload, download,
 525 organize, and share data within that allocated space.

526 Petrel uses Globus to implement the MRDP model on top of storage provided by Argonne. It
 527 implements a simple workflow (based on Globus Groups) for users to request an allocation. It then,
 528 using code similar to Listing 1, creates a directory for the allocation and a shared endpoint to manage the
 529 allocation, and assigns the requesting user as the "manager" of that shared endpoint. The user can then
 530 manage their allocation as if it were storage on their personal resources, uploading and downloading data,

531 and selecting who may access (read/write) paths in that storage.

532 Scalable data publication

533 Globus data publication (Chard et al., 2015) enables researchers to publish and access data in user-
534 managed collections. Following the MRDP design pattern, the system is implemented as a cloud-hosted
535 service with data storage provided by decoupled Globus endpoints. Users define their own publication
536 collections by specifying a Globus endpoint on which data are to be stored. The publication service then
537 manages the workflows for submitting a dataset, associating a persistent identifier with the dataset, and
538 recording user-supplied metadata. Permitted users can then publish data in the collection by following
539 this workflow. Once published, other users are then able to discover and download published data.

540 The service uses Globus transfer to manage data. Upon submission of a new dataset, the publication
541 service creates a unique directory in the collection's shared endpoint and shares it (with write access)
542 to the submitting user. After the submission is complete, the service removes the write permission (in
543 effect making the endpoint immutable) and proceeds through the submission and curation workflows, by
544 changing permissions on the directory. When the submission is approved for publication, the service
545 modifies permissions to match the collection's policies. It now acts as a typical MRDP, providing access
546 to published data for discovery and download.

547 Data delivery at Advanced Photon Source

548 The Advanced Photon Source (APS) at Argonne National Laboratory, like many experimental facilities
549 worldwide, serves thousands of researchers every year, many of whom collect data and return to their
550 home institution. In the past, data produced during an experiment was invariably carried back on physical
551 media. However, as data sizes have grown and experiments have become more collaborative, that approach
552 has become less effective. Data transfer via network is preferred; the challenge is to integrate data transfer
553 into the experimental workflow of the facility in a way that is automated, secure, reliable, and scalable.

554 The DMagic system De Carlo (2017) does just that. DMagic integrates with APS administrative and
555 facility systems to deliver data to experimental users. Before the experiment begins, it creates a shared
556 endpoint on a large storage system maintained by Argonne's computing facility. DMagic then retrieves
557 the list of approved users for the experiment and adds permissions for those users to the shared endpoint.
558 It then monitors the experiment data directory at the APS experimental facility and copies new files
559 automatically to that shared endpoint, from which they can be retrieved by any approved user.

560 EVALUATION OF MRDP ADOPTION

561 The advantages of using high performance data transfer protocols over traditional protocols (e.g., HTTP,
562 SCP) have been well explored (Mattmann et al., 2006; Rao et al., 2016; Subramoni et al., 2010). The
563 benefits of building upon a professionally managed platform are also well established (Cusumano, 2010).
564 Thus, rather than comparing MRDP performance with that of legacy approaches, we examine adoption of
565 the MRDP pattern in real-world deployments. Specifically, we explore usage of five MRDPs: 1) NCAR's
566 Research Data Archive; 2) the Sanger imputation service; 3) Petrel data sharing service; 4) Globus data
567 publication (Publish); and 5) the Cornell Advanced Computing (CAC) Archive.

568 Table 1 presents for each portal the number of transfers, number of unique users, total amount of data
569 transferred, and average transfer rates. Figure 7 and Figure 8 show the total data transferred per day and
570 file size vs. transfer rate for individual transfers, respectively, for three of these MRDPs. These results
571 highlight the large amounts of data that can be handled by such portals. In fact, all but one moves in
572 excess of 1 TB per day. (The peak transfer volume of more than 1 PB recorded for Petrel arises because
573 transfer volumes are assigned to the day on which a transfer completes.) These graphs also highlight the
574 wide range of MRDP characteristics that we encounter "in the wild": for example, whether source or sink,
575 the scales of data handled, and the network bandwidths available.

576 We comment on a few specifics of the different portals shown in the figures. The Sanger portal accepts
577 user-uploaded datasets and returns processed datasets that are roughly an order of magnitude larger. Its
578 average transfer rates are lower than for the other portals as it is used by individual users with varied
579 connectivity.

580 Petrel handles extremely large data and provides high-speed data access via its many DTNs and
581 high-speed networks that connect it to users within Argonne and at other labs.

Table 1. Usage summary for five MRDP deployments: total operations, unique users, total data, and average transfer rate, each for both outgoing and incoming transfers.

Name	Operations		Users		Data (TB)		Rate (mbps)	
	Out	In	Out	In	Out	In	Out	In
RDA	6,677	84	399	1	596.6	0.0	16.5	1.1
Sanger	5,667	4,757	328	346	298.9	49.6	25.6	13.6
Petrel	35,941	18,015	225	100	1,630.0	3,320.0	110.7	213.4
Publish	3,563	3,243	124	84	8.5	7.9	3.8	30.1
CAC	477	3,190	30	49	359.5	705.3	173.2	37.5

582 The RDA portal primarily serves data of modest size (10s of GBs) and at modest rates to many
 583 endpoints (likely as consumers are using a variety of destinations). Nevertheless, it can achieve high
 584 performance when a destination endpoint has suitable capacity, as is shown in Figure 8b, which extracts
 585 from Figure 8a the transfers from RDA to the National Energy Research Supercomputing Center (NERSC),
 586 a high performance computing center in Berkeley, California, with a substantial Globus deployment.

587 Globus data publication is a unique portal in that its storage is distributed across 300 different Globus
 588 endpoints. Its transfer rate is therefore variable and usage is sporadic.

589 Finally, CAC is used by institutional users as an archival service. As such, it is primarily used as a
 590 destination, and transfer rates are high as users share a high-speed internal network.

591 RELATED WORK

592 The MRDP design pattern that we have presented here codifies the experience of many groups who have
 593 deployed and applied research data portals. Our discussion of implementation techniques reflects this
 594 experience. However, the design pattern could also be implemented with other technologies, as we now
 595 discuss.

596 The Globus transfer service typically employs the Globus implementation of the GridFTP transfer pro-
 597 tocol (Allcock et al., 2005) for file transfers, which uses techniques such as multiple TCP channels (Hacker
 598 et al., 2004) to optimize transfer performance. Other protocols and tools that can be used for this purpose
 599 include Fast TCP (Jin et al., 2005), Scalable TCP (Kelly, 2003), UDT (Gu and Grossman, 2007), and
 600 bbcp (Hanushevsky et al., 2001).

601 PhEDEx (Egeland et al., 2010) and the CERN File Transfer Service (Laure et al., 2006) are two
 602 examples of services that support management of file transfers. Both are designed for use in high energy
 603 physics experiments. The EUDAT (Lecarpentier et al., 2013) project's B2STAGE service supports
 604 the transfer, via the GridFTP protocol, of research data between EUDAT storage resources and high-
 605 performance computing (HPC) workspaces. None of these systems provide APIs for integration with data
 606 portals and Science DMZs.

607 Apache Airavata (Pierce et al., 2015) provides general-purpose middleware for building science
 608 gateways, but does not address data movement or integration with Science DMZs. CyVerse (Goff et al.,
 609 2011) provides a scalable data store, built on iRODS (Rajasekar et al., 2010), and specialized APIs (Dooley
 610 et al., 2012) to access its storage services. It supports Cyberduck, amongst other tools, for transferring
 611 data using an array of protocols (e.g., HTTP, FTP, WebDAV) directly to Windows and Mac PCs. Thus,
 612 while it may separate control and data channels, it does not support third-party or high performance data
 613 transfer nor does it integrate with Science DMZs and DTNs.

614 Many services support scientific data publication, including Dataverse (Crosas, 2011), figshare, and
 615 Zenodo. Increasingly, these services are incorporating support for large amounts of data. For example,
 616 the developers of the Structural Biology Data Grid (Morin et al., 2013) are working to integrate Globus
 617 with Dataverse (Meyer et al., 2016). Similarly, publication repositories such as figshare now support data
 618 storage on institutional or cloud storage, thereby achieving some separation of control and data channels.
 619 While these implementations follow some aspects of the MRDP design pattern, they do not integrate with
 620 Science DMZs and their implementations are proprietary and cannot be easily generalized and adopted by
 621 others.

622 The MRDP design pattern could also be implemented using commercial cloud services, such as
 623 those provided by Amazon, Google, and Microsoft. Indeed, many public scientific datasets are now

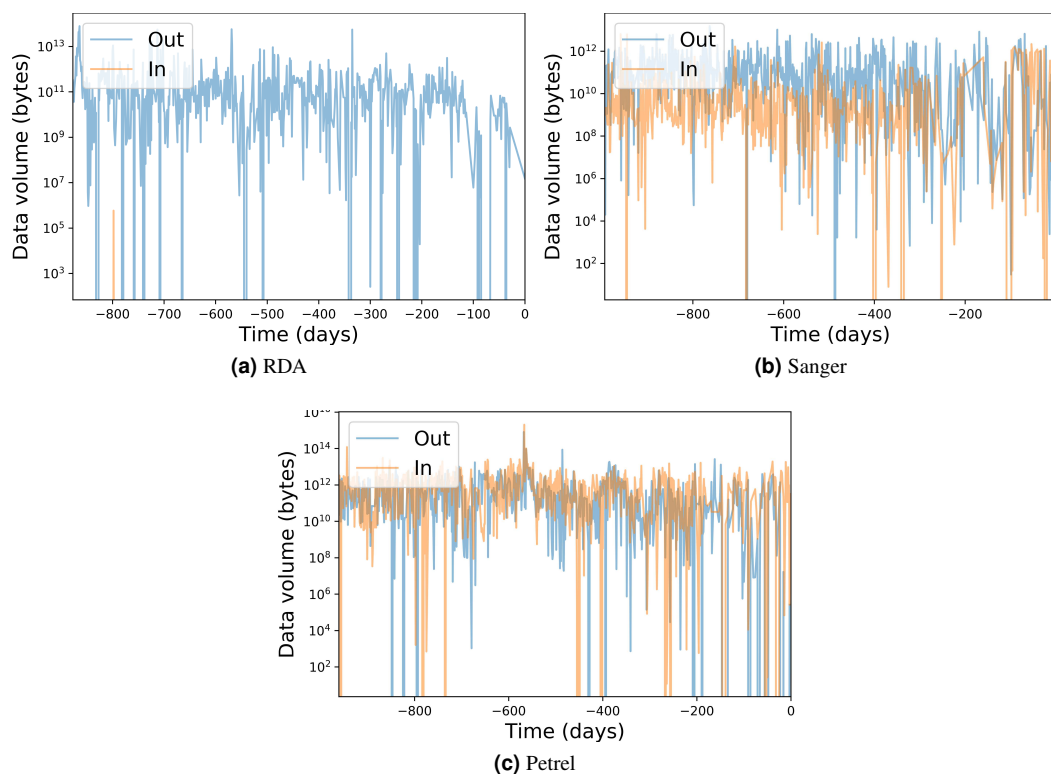


Figure 7. Input/output transfer volumes per day for three MRDP instances. The numbers on the x axis represent days prior to March 9, 2017.

624 hosted on cloud platforms and researchers are increasingly leveraging cloud services for managing and
 625 analyzing data (Babuji et al., 2016). Cloud services provide scalable data management, rich identity
 626 management, and secure authentication and authorization, all accessible via APIs—a complete set of
 627 capabilities needed to implement the MRDP design pattern. However, each cloud provider is a walled
 628 garden, not easily integrated with other systems, and cloud provider business models require that MRDP
 629 administrators develop methods to recoup service costs related to data storage and egress. In contrast, our
 630 MRDP reference implementation allows data to be stored and retrieved from many locations.

631 SUMMARY

632 We have described the state of the practice for delivering scientific data through what we call the modern
 633 research data portal (MRDP) design pattern. The MRDP pattern addresses the shortcomings of the
 634 monolithic legacy research data portal by improving transfer performance, security, reliability, and ease of
 635 implementation while also reducing operational complexity.

636 We have shown how high performance networks and cloud service APIs can be used to create
 637 performant and particularly simple implementations of this design pattern. In a typical deployment, as
 638 shown in Figure 3, the control logic of Listing 1 runs on a protected computer behind the institutional
 639 firewall, to protect that sensitive logic against attack, while the storage system(s) on which data reside sit
 640 inside the Science DMZ, with Globus endpoint(s) deployed on DTNs for high-speed access. The control
 641 logic makes REST API calls to the Globus cloud service to create shared endpoints, transfer files, manage
 642 permissions, and so forth.

643 The sample code we have presented shows how developers can automate data management tasks by
 644 using modern cloud-based data services to create powerful research data portals (accessible via Web,
 645 mobile, custom applications, and command line) that leverage Science DMZ paths for data distribution,
 646 staging, replication, and other purposes. In our example we leverage Globus to outsource all identity
 647 management and authentication functions. The MRDP needs simply to provide service-specific au-

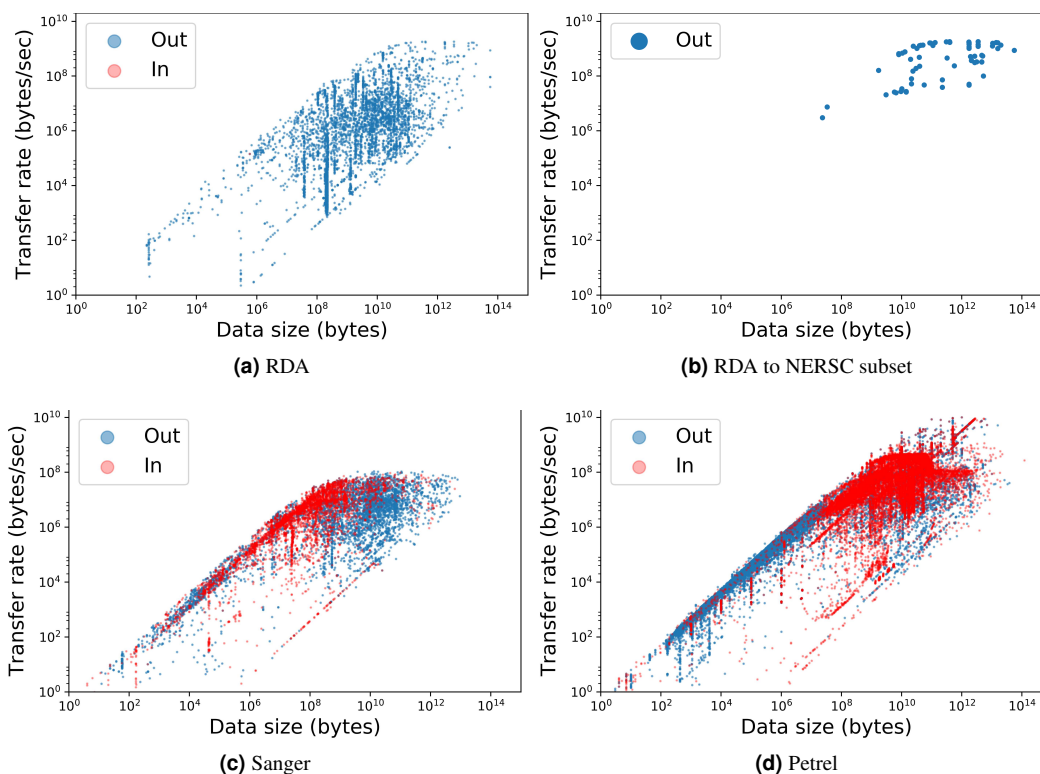


Figure 8. Transfer rate vs. data size for MRDP instances. The two upper figures show (a) all transfers for the RDA instance and (b) only transfers from RDA to NERSC, with points in a larger size, as there are fewer of them. The two lower figures show all transfers for the Sanger and Petrel instances, respectively. Each point in each scatter plot represents a single transfer request, which may involve many files. Incoming and outgoing transfers are distinguished; RDA has only one incoming transfer.

648 thorization, which can be performed on the basis of identity or group membership. And because all
 649 interactions are compliant with OAuth 2 and OpenID Connect standards, any application that speaks
 650 these protocols can use the MRDP service like they would any other; the MRDP service can seamlessly
 651 leverage other services; and other services can leverage MRDP service. The use of best-practice and
 652 standards-compliant implementations for data movement, automation, authentication, and authorization is
 653 a powerful combination.

654 The benefits of the MRDP approach lie not only in the separation of concerns between control logic
 655 and data movement. In addition, the data portal developer and admin both benefit from the ability to hand
 656 off the management of file access and transfers to the Globus service. In the last three years we have
 657 observed steady adoption of the MRDP design pattern. We described five illustrative implementations
 658 that variously serve research data, support analysis of uploaded data, provide flexible data sharing, enable
 659 data publication, and facilitate data archival. Collectively, these deployments have performed more than
 660 80,000 transfers totaling almost seven petabytes over the past three years.

661 ACKNOWLEDGMENTS

662 We thank the Globus team for the development of the technologies described here, and participants in
 663 “Building the Modern Research Data Portal” workshops for their feedback. This work was supported in
 664 part by NSF grant ACI-1148484 and by the DOE under contract DE-AC02-06CH11357.

665 REFERENCES

666 Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I. (2005). The
 667 Globus striped GridFTP framework and server. In *ACM/IEEE Conference on Supercomputing*, page 54.

- 668 Babuji, Y. N., Chard, K., Gerow, A., and Duede, E. (2016). Cloud Kotta: Enabling secure and scalable
669 data analytics in the cloud. In *IEEE International Conference on Big Data*, pages 302–310.
- 670 Barnett, W., Welch, V., Walsh, A., and Stewart, C. A. (2011). A roadmap for using NSF cyberinfrastructure
671 with InCommon. <http://hdl.handle.net/2022/13024>.
- 672 Basney, J. and Gaynor, J. (2011). An OAuth service for issuing certificates to science gateways for
673 TeraGrid users. In *TeraGrid Conference: Extreme Digital Discovery*, page 32. ACM.
- 674 Berners-Lee, T. (1989). Information management: A proposal. [https://www.w3.org/History/
675 1989/proposal.html](https://www.w3.org/History/1989/proposal.html).
- 676 Borgman, C. L. (2012). The conundrum of sharing research data. *Journal of the American Society for
677 Information Science and Technology*, 63(6):1059–1078.
- 678 Chard, K., Lidman, M., McCollam, B., Bryan, J., Ananthkrishnan, R., Tuecke, S., and Foster, I. (2016).
679 Globus nexus: A platform-as-a-service provider of research identity, profile, and group management.
680 *Future Generation Computer Systems*, 56:571 – 583.
- 681 Chard, K., Pruyne, J., Blaiszik, B., Ananthkrishnan, R., Tuecke, S., and Foster, I. (2015). Globus
682 data publication as a service: Lowering barriers to reproducible science. In *11th IEEE International
683 Conference on e-Science (e-Science)*, pages 401–410.
- 684 Chard, K., Tuecke, S., and Foster, I. (2014). Efficient and secure transfer, synchronization, and sharing of
685 big data. *IEEE Cloud Computing*, 1(3):46–55.
- 686 Crosas, M. (2011). The Dataverse network: An open-source application for sharing, discovering and
687 preserving data. *D-lib Magazine*, 17(2).
- 688 Cusumano, M. (2010). Cloud computing and SaaS as new computing platforms. *Communications of the
689 ACM*, 53(4):27–29.
- 690 Dart, E., Rotman, L., Tierney, B., Hester, M., and Zurawski, J. (2013). The Science DMZ: A network
691 design pattern for data-intensive science. In *International Conference on High Performance Computing,
692 Networking, Storage and Analysis, SC '13*, pages 85:1–85:10, New York, NY, USA. ACM.
- 693 De Carlo, F. (2017). DMagic data management system. <http://dmagic.readthedocs.io>.
- 694 Dooley, R., Vaughn, M., Stanzione, D., Terry, S., and Skidmore, E. (2012). Software-as-a-service: the
695 iPlant foundation API. In *5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers*.
- 696 Egeland, R., Wildishb, T., and Huang, C.-H. (2010). PhEDEx data service. *Journal of Physics: Conference
697 Series*, 219.
- 698 ESnet (2017). ESnet Fasterdata Knowledge Base. <http://fasterdata.es.net>.
- 699 Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable
700 Object-oriented Software*. Addison-Wesley.
- 701 Goff, S. A., Vaughn, M., McKay, S., Lyons, E., Stapleton, A. E., Gessler, D., Matasci, N., Wang, L.,
702 Hanlon, M., Lenards, A., Muir, A., Merchant, N., Lowry, S., Mock, S., Helmke, M., Kubach, A., Narro,
703 M., Hopkins, N., Micklos, D., Hilgert, U., Gonzales, M., Jordan, C., Skidmore, E., Dooley, R., Cazes,
704 J., McLay, R., Lu, Z., Pasternak, S., Koesterke, L., Piel, W. H., Grene, R., Noutsos, C., Gendler, K.,
705 Feng, X., Tang, C., Lent, M., Kim, S.-J., Kvilekval, K., Manjunath, B. S., Tannen, V., Stamatakis, A.,
706 Sanderson, M., Welch, S. M., Cranston, K. A., Soltis, P., Soltis, D., O'Meara, B., Ane, C., Brutnell, T.,
707 Kleibenstein, D. J., White, J. W., Leebens-Mack, J., Donoghue, M. J., Spalding, E. P., Vision, T. J.,
708 Myers, C. R., Lowenthal, D., Enquist, B. J., Boyle, B., Akoglu, A., Andrews, G., Ram, S., Ware, D.,
709 Stein, L., and Stanzione, D. (2011). The iPlant collaborative: Cyberinfrastructure for plant biology.
710 *Frontiers in Plant Science*, 2:34.
- 711 Gu, Y. and Grossman, R. L. (2007). UDT: UDP-based data transfer for high-speed wide area networks.
712 *Comput. Netw.*, 51(7):1777–1799.
- 713 Hacker, T. J., Noble, B. D., and Athey, B. D. (2004). Improving throughput and maintaining fairness
714 using parallel TCP. In *IEEE InfoCom*.
- 715 Hanemann, A., Boote, J. W., Boyd, E. L., Durand, J., Kudarimoti, L., Łapacz, R., Swany, D. M., Trocha,
716 S., and Zurawski, J. (2005). perfSONAR: A service oriented architecture for multi-domain network
717 monitoring. In *International Conference on Service-Oriented Computing*, pages 241–254. Springer.
- 718 Hanushevsky, A., Trunov, A., and Cottrell, L. (2001). Peer to peer computing for secure high performance
719 data copying. Technical Report SLAC-PUB-8908, Stanford Linear Accelerator Center.
- 720 Hardt, D. (2012). OAuth 2.0 authorization framework specification. [http://tools.ietf.org/
721 html/rfc6749](http://tools.ietf.org/html/rfc6749).
- 722 Hey, T., Tansley, S., and Tolle, K. (2009). The fourth paradigm. *Microsoft Research*.

- 723 Jin, C., Wei, D., Low, S. H., Bunn, J., Choe, H. D., Doyle, J. C., Newman, H., Ravot, S., Singh, S., and
724 Paganini, F. (2005). FAST TCP: From theory to experiments. *IEEE Network*, 19(1):4–11.
- 725 Kelly, T. (2003). Scalable TCP: Improving performance in highspeed wide area networks. *ACM*
726 *SIGCOMM Computer Communication Review*, 33(2):83–91.
- 727 Klimeck, G., McLennan, M., Brophy, S. P., Adams III, G. B., and Lundstrom, M. S. (2008). nanohub.org:
728 Advancing education and research in nanotechnology. *Computing in Science & Engineering*, 10(5):17–
729 23.
- 730 Laure, E., Fisher, S., Frohner, A., Grandi, C., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F.,
731 White, J., Barroso, M., Buncic, P., Hemmer, F., Di Meglio, A., and Edlund, A. (2006). Programming the
732 grid with gLite. Technical Report EGEE-TR-2006-001, CERN. [http://cds.cern.ch/record/
733 936685](http://cds.cern.ch/record/936685).
- 734 Lawrence, K. A., Zentner, M., Wilkins-Diehr, N., Wernert, J. A., Pierce, M., Marru, S., and Michael, S.
735 (2015). Science gateways today and tomorrow: Positive perspectives of nearly 5000 members of the
736 research community. *Concurrency and Computation: Practice and Experience*, 27(16):4252–4268.
- 737 Lecarpentier, D., Wittenburg, P., Elbers, W., Michelini, A., Kanso, R., Coveney, P., and Baxter, R. (2013).
738 EUDAT: A new cross-disciplinary data infrastructure for science. *International Journal of Digital*
739 *Curation*, 8(1):279–287.
- 740 Mattmann, C. A., Kelly, S., Crichton, D. J., Hughes, J. S., Hardman, S., Ramirez, P., and Joyner, R. (2006).
741 A classification and evaluation of data movement technologies for the delivery of highly voluminous
742 scientific data products. Technical report, Jet Propulsion Laboratory.
- 743 McCarthy, S., Das, S., Kretzschmar, W., Delaneau, O., Wood, A. R., Teumer, A., Kang, H. M., Fuchsberger,
744 C., Danecek, P., Sharp, K., Luo, Y., Sidore, C., Kwong, A., Timpson, N., Koskinen, S., Vrieze, S.,
745 Scott, L. J., Zhang, H., Mahajan, A., Veldink, J., Peters, U., Pato, C., van Duijn, C. M., Gillies, C. E.,
746 Gandin, I., Mezzavilla, M., Gilly, A., Cocca, M., Traglia, M., Angius, A., Barrett, J. C., Boomsma, D.,
747 Branham, K., Breen, G., Brummett, C. M., Busonero, F., Campbell, H., Chan, A., Chen, S., Chew,
748 E., Collins, F. S., Corbin, L. J., Smith, G. D., Dedoussis, G., Dorr, M., Farmaki, A.-E., Ferrucci, L.,
749 Forer, L., Fraser, R. M., Gabriel, S., Levy, S., Groop, L., Harrison, T., Hattersley, A., Holmen, O. L.,
750 Hveem, K., Kretzler, M., Lee, J. C., McGue, M., Meitinger, T., Melzer, D., Min, J. L., Mohlke, K. L.,
751 Vincent, J. B., Nauck, M., Nickerson, D., Palotie, A., Pato, M., Pirastu, N., McInnis, M., Richards,
752 J. B., Sala, C., Salomaa, V., Schlessinger, D., Schoenherr, S., Slagboom, P. E., Small, K., Spector, T.,
753 Stambolian, D., Tuke, M., Tuomilehto, J., den Berg, L. H. V., Rheenen, W. V., Volker, U., Wijmenga,
754 C., Toniolo, D., Zeggini, E., Gasparini, P., Sampson, M. G., Wilson, J. F., Frayling, T., de Bakker, P.
755 I. W., Swertz, M. A., McCarroll, S., Kooperberg, C., Dekker, A., Altshuler, D., Willer, C., Iacono, W.,
756 Ripatti, S., Soranzo, N., Walter, K., Swaroop, A., Cucca, F., Anderson, C. A., Myers, R. M., Boehnke,
757 M., McCarthy, M. I., Durbin, R., Abecasis, G., and Marchini, J. (2016). A reference panel of 64,976
758 haplotypes for genotype imputation. *Nature genetics*, 48(10):1279–1283.
- 759 McLennan, M. and Kennell, R. (2010). HUBzero: A platform for dissemination and collaboration in
760 computational science and engineering. *Computing in Science & Engineering*, 12(2).
- 761 Meyer, P. A., Socias, S., Key, J., Ransey, E., Tjon, E. C., Buschiazzo, A., Lei, M., Botka, C., Withrow,
762 J., Neau, D., Rajashankar, K., Anderson, K. S., Baxter, R. H., Blacklow, S. C., Boggon, T. J., Bonvin,
763 A. M. J. J., Borek, D., Brett, T. J., Caffisch, A., Chang, C.-I., Chazin, W. J., Corbett, K. D., Cosgrove,
764 M. S., Crosson, S., Dhe-Paganon, S., Cera, E. D., Drennan, C. L., Eck, M. J., Eichman, B. F., Fan,
765 Q. R., Ferré-D’Amaré, A. R., Fromme, J. C., Garcia, K. C., Gaudet, R., Gong, P., Harrison, S. C.,
766 Heldwein, E. E., Jia, Z., Keenan, R. J., Kruse, A. C., Kvangsakul, M., McLellan, J. S., Modis, Y., Nam,
767 Y., Otwinowski, Z., Pai, E. F., Pereira, P. J. B., Petosa, C., Raman, C. S., Rapoport, T. A., Roll-Mecak,
768 A., Rosen, M. K., Rudenko, G., Schlessinger, J., Schwartz, T. U., Shamoo, Y., Sondermann, H., Tao,
769 Y. J., Tolia, N. H., Tsodikov, O. V., Westover, K. D., Wu, H., Foster, I., Fraser, J. S., Maia, F. R. N. C.,
770 Gonen, T., Kirchhausen, T., Diederichs, K., Crosas, M., and Sliz, P. (2016). Data publication with the
771 Structural Biology Data Grid supports live analysis. *Nature Communications*, 7.
- 772 Morin, A., Eisenbraun, B., Key, J., Sanschagrin, P. C., Timony, M. A., Ottaviano, M., and Sliz, P. (2013).
773 Collaboration gets the most out of software. *eLife*, 2:e01456.
- 774 Pierce, M. E., Marru, S., Gunathilake, L., Wijeratne, D. K., Singh, R., Wimalasena, C., Ratnayaka, S., and
775 Pamidighantam, S. (2015). Apache Airavata: Design and directions of a science gateway framework.
776 *Concurrency and Computation: Practice and Experience*, 27(16):4282–4291.
- 777 Rajasekar, A., Moore, R., Hou, C.-y., Lee, C. A., Marciano, R., de Torcy, A., Wan, M., Schroeder, W.,

- 778 Chen, S.-Y., Gilbert, L., Tooby, P., and Zhu, B. (2010). iRODS Primer: Integrated rule-oriented data
779 system. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 2(1):1–143.
- 780 Rao, N. S., Liu, Q., Sen, S., Hinkel, G., Imam, N., Foster, I., Kettimuthu, R., Settlemeyer, B. W., Wu, C. Q.,
781 and Yun, D. (2016). Experimental analysis of file transfer rates over wide-area dedicated connections.
782 In *IEEE 18th International Conference on High Performance Computing and Communications*, pages
783 198–205. IEEE.
- 784 Russell, M., Allen, G., Daues, G., Foster, I., Seidel, E., Novotny, J., Shalf, J., and Von Laszewski, G.
785 (2001). The Astrophysics Simulation Collaboratory: A science portal enabling community software
786 development. In *10th IEEE International Symposium on High Performance Distributed Computing*,
787 pages 207–215.
- 788 Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and Mortimore, C. (2014). OpenID Connect Core
789 1.0. http://openid.net/specs/openid-connect-core-1_0.html.
- 790 Subramoni, H., Lai, P., Kettimuthu, R., and Panda, D. K. (2010). High performance data transfer in grid
791 environment using GridFTP over InfiniBand. In *10th IEEE/ACM International Conference on Cluster,
792 Cloud and Grid Computing*, pages 557–564. IEEE.
- 793 Tenopir, C., Allard, S., Douglass, K., Aydinoglu, A. U., Wu, L., Read, E., Manoff, M., and Frame, M.
794 (2011). Data sharing by scientists: Practices and perceptions. *PLOS ONE*, 6(6):e21101.
- 795 Tuecke, S., Ananthakrishnan, R., Chard, K., Lidman, M., McCollam, B., and Foster, I. (2015). Globus
796 Auth: A research identity and access management platform. In *12th IEEE International Conference on
797 e-Science (e-Science)*.
- 798 Wilkins-Diehr, N., Gannon, D., Klimeck, G., Oster, S., and Pamidighantam, S. (2008). TeraGrid science
799 gateways and their impact on science. *Computer*, 41(11).