I'll Take That to Go: Big Data Bags and Minimal Identifiers for Exchange of Large, Complex Datasets

Kyle Chard[†], Mike D'Arcy[*], Eric W. Deutsch[‡], Ian Foster[†],
Carole Goble[§], Ben Heavner[‡], Carl Kesselman[*], Ravi Madduri[†], Stian Soiland-Reyes[§]
[†]*Computation Institute, The University of Chicago, Chicago IL, USA*
[*]*Information Sciences Institute, University of Southern California, Marina del Rey CA, USA*
[‡]*Institute for Systems Biology, Seattle, WA, USA*
[§]*School of Computer Science, The University of Manchester, Manchester, UK*

***Abstract*—Big data workflows often require the assembly and exchange of complex, multi-element *datasets*. For example, in biomedical applications, the input to an analytic pipeline can be a dataset consisting thousands of images and genome sequences assembled from diverse repositories, requiring a description of the contents of the dataset in a concise and unambiguous form. Typical approaches to creating datasets for big data workflows assume that all data reside in a single location, requiring costly data marshaling and permitting errors of omission and commission because dataset members are not explicitly specified. We address these issues by proposing simple methods and tools for assembling, sharing, and analyzing large and complex datasets that scientists can easily integrate into their daily workflows. These tools combine a simple and robust method for describing data collections (BDBags), data descriptions (Research Objects), and simple persistent identifiers (Minids) to create a powerful ecosystem of tools and services for big data analysis and sharing. We present these tools and use biomedical case studies to illustrate their use for the rapid assembly, sharing, and analysis of large datasets.**

I. Introduction

Many domains of science must frequently manage, share, and analyze complex, large data collections—what we call *datasets* in this paper—that comprise many, often large, files of different types. Biomedical applications are a case in point [1]. For example, an imaging genetics study may encompass thousands of high-resolution 3D images, terabytes in size; whole genome sequences, each tens of gigabytes in size; and other heterogeneous clinical data.

Data management and analysis tools typically assume that the data are assembled in one location. However, this assumption is often not well founded in the case of big data. Due to their size, complexity and diverse methods of production, files in a dataset may be distributed across multiple storage systems: for example, multiple imaging and genomics repositories. As a result, apparently routine data manipulation workflows become rife with mundane complexities as researchers struggle to assemble large, complex datasets; position data for access to analytic tools; document and disseminate large output datasets; and track the inputs and outputs to analyses for purpose of reproducibility [2].

Researchers thus need robust mechanisms for describing, referring to, and exchanging large and complex datasets. However, while sophisticated conventions have been defined for encoding data and associated metadata, and persistent identifier schemes developed that can bind rich sets of metadata to carefully curated data, all impose significant overheads on the researcher and thus are often not used in practice. We address this shortcoming here by proposing simple yet powerful mechanisms for specifying, sharing, and managing complex, distributed, large datasets.

This paper makes three principal contributions. First, we define new methods, based on the integration of existing and new mechanisms, for organizing, naming, and describing big datasets for purposes of exchange. Second, we describe efficient implementations of these methods, based on a mix of online services and simple downloadable code. Third, we demonstrate via a range of applications that these methods simplify complex research processes and encourage reproducible research.

The rest of this paper is as follows. In §II, we present motivation and requirements. We next describe our approach to defining (§III), describing (§IV), and robustly identifying (§V) complex distributed big datasets. In §VI we present tools to support researcher workflows and applications. In §VII we describe the application of our ecosystem to biomedical big data problems. Finally, in §VIII and §IX, we review related work and present our conclusions.

II. Technical approach

We identify six requirements for tools to support the creation and exchange of complex, big, data collections (*datasets*) made up of many directories and files (*elements*).

1. **Enumeration**: The dataset's elements must be explicitly enumerated, so that subsequent additions or deletions can be detected. Thus we cannot, for example, simply define a dataset as "the contents of directory D."
2. **Fixity**: Scientific repeatability requires a robust way of verifying that we have the intended versions of dataset contents, so that data consumers can detect errors in data transmission or modifications to data elements.

3. **Description**: We need interoperable methods for tracking the attributes (metadata) and origins (provenance) of dataset contents.
4. **Identification**: We require a reliable and concise way of referring to datasets for purposes of collaboration, publication, and citation.
5. **Distribution**: A dataset is able to contain elements from more than one location.
6. **Simplicity**: Methods should not impose significant user overhead or require that complex software be deployed on researcher computers.

Requirements 1–3 address the need to make the dataset a basic building block for scientific repeatability. **Enumeration** and **Fixity** ensure that a dataset has exactly the data elements intended by the creator, and that each data element has not been modified from the time the dataset was created. **Description** facilitates use of a dataset by a consumer, whether a collaborator or analysis program. **Identification** and **Distribution** streamline the creation, exchange, and discovery of datasets by reducing the work required to create a dataset, and allowing dataset exchange via the simple transfer of a concise, well-defined, and invariant dataset name. **Simplicity** is directed towards usability. If creation of big datasets is significantly more complex than the status quo of directories of files, then they will not be used. Solutions that meet these requirements facilitate broad goals of scientific repeatability and data reuse, by making data findable, accessible, interoperable, and reusable (FAIR) [3].

To meet these requirements we have developed a solution that has three main components:

- The BDBag: A mechanism for *defining* a dataset and its contents by enumerating its elements, regardless of their location (enumeration, fixity and distribution)
- The Research Object (RO): A means for *characterizing* a dataset and its contents with arbitrary levels of detail, regardless of their location (description);
- The Minid: A method for uniquely *identifying* a dataset and, if desired, its constituent elements, regardless of their location (identify, fixity)

A BDBag specifies the exact contents of a dataset, while ROs describe those contents for purposes of reuse. The dataset or any file within the dataset is unambiguously named by a persistent, actionable identifier: the Minid. By using lightweight approaches we have rapidly assembled the all important ecosystem of tools for exchanging, creating, and consuming datasets created using these methods for defining, naming, and describing them.

We next describe the BDBag, RO, and Minid in turn.

III. BDBags: Representing Distributed Datasets

We first consider how to unambiguously specify the data elements that make up a dataset. We assume that these data elements are all represented as *files*: named sequences of digital data. We make no assumptions about the coding of file contents: If a user wishes to include structured data such as a database in a dataset, the data must first be exported in some file format, such as a database dump or comma separated value (CSV) file.

We require a dataset representation that allow a dataset's contents to be distributed over multiple locations, as in the case of a biomedical research dataset that encompasses patient data from several repositories. To minimize the overheads associated with using a dataset, we require further that it be possible to validate and access a dataset's contents and to interpret some minimal metadata, all without access to tools other than those available in a standard operating system deployment. This latter requirement argues against binary representations.

Existing formats for encoding complex data, such as HDF and NetCDF, require complex tooling to consume. They also require significant up front design to specialize the container formats to a specific use case or domain. Archive formats, such as ZIP, TAR, and variants, are commonly used for smaller datasets. These formats use a simple hierarchical naming model (i.e., directory structure) to organize data elements. The tooling to create and unpack such archives is ubiquitous, and unpacking creates a file system representation of the archive that can be manipulated and examined without additional software. However, these archive formats do not define mechanisms for determining whether an intended element is missing or if the content has been modified. Nor do they provide mechanisms for referencing distributed elements: all data to be included in the dataset must be assembled in one place.

These requirements led us to define the BDBag specification for describing large, distributed datasets. We next review the BagIt specification that BDBag extends and introduce the BDBag extensions.

A. The BagIt specification

The BagIt specification [4], originally developed in the digital library community, implements the "enclose and deposit" (also "bag it and tag it": hence BagIt) approach [5] to data aggregation and description. The specification requires that a bag comprise a collection of data files plus a *file manifest*. To enable validation of the dataset's completeness and correctness, the manifest must provide a checksum for each file. The specification also defines the notion of a *partial bag*, a bag in which the contents of some files listed in the manifest are omitted from the bag contents, but are instead specified by a reference to a location from which the content may be retrieved.

The BagIt specification defines a directory structure and required and optional files within that structure, as well as a set of methods to serialize the directory into an established archive format, such as ZIP. As shown in Figure 1, data files in a bag are located in a payload directory named

`data`, within which files may be hierarchically organized using subdirectories. In order to make the contents of the bag explicit, a manifest file must be provided that lists the files in the payload data directory along with a checksum for each file. Alternative checksum algorithms can be used, with the algorithm indicated in the manifest file name, e.g., `manifest-md5.txt` or `manifest-sha256.txt`. The manifest file must appear at the same level in the directory as the `data` directory. This information means that software need simply to examine the manifest file to detect accidentally removed or corrupted files.

A bag must also contain a file `bagit.txt` with contents specifying the BagIt specification version number and the character encoding used for any tag files. It may also include one or more *tag files* containing metadata describing payload elements. One important tag file, `fetch.txt` must be included if any files listed in the manifest are not in the payload directory. If files are missing, this file then contains a line of the form (URL, LENGTH, FILENAME) for each missing file to indicate its name and a URL from which it may be obtained. (The URL can indicate the protocol to be used: e.g., HTTP or GridFTP). Once retrieved, the file can be placed in the appropriate location in the data directory, and its checksum validated from the manifest file.

Finally, a bag must contain a tag manifest file, `tag-md5.txt` and/or `tag-sha256.txt`, enumerating the name and checksum of `bagit.txt` and any tag files. This information allows software to validate that these files are present and correct.

B. The BDBag BagIt Profile

The BagIt specification allows for the definition of bag *profiles* to address the needs of specific use cases by imposing a set of conventions using various optional aspects of the BagIt specification. A BagIt profile is instantiated as an optional element of the `bag-info.txt` metadata file, which is referenced by a single field named `BagIt-Profile-Identifier`, and whose value is an HTTP URI referring to a JSON formatted file that describes the constraints to impose on the bag structure.

The BDBag BagIt profile uses this profiling mechanism to address certain limitations that we encountered when using BagIt to deal with big data. Importantly, our use of a profile to define our additional requirements means that every BDBag is also a bag.

Figure 1 shows an example of a BDBag, highlighting the bag elements that are specific or specialized to the BDBag BagIt profile specification. To enable rapid evaluation of a bag's completeness (but not necessarily its validity) by bag processing tools, the BDBag BagIt profile requires a `Payload-Oxum` field in `bag-info.txt` to document the number of bytes and files in the payload, as defined in the BagIt specification [4].

`examplebag/`	Top level name
`bag-info.txt`	Metadata for the bag
`bagit.txt`	BagIt version, encoding info
`data/`	The BDBag's contents:
`mydirectory/`	User directory
`file1`	A first user file
`file2`	A second user file
`fetch.txt`	How to fetch missing elements
`manifest-md5.txt`	MD5 checksums for data files
`manifest-sha256.txt`	SHA checksums for data files
`metadata/`	Tag directory for RO metadata
`manifest.json`	RO metadata as JSON-LD
`tagmanifest-md5.txt`	MD5 checksum for tags
`tagmanifest-sha256.txt`	SHA checksum for tags

*Figure 1: An example BDBag. Elements added or specialized for BDBag are in **boldface***

To enable description of BDBag contents, the BDBag BagIt profile requires a *RO manifest* (see §IV) as a tagfile at the bag-root relative path `metadata/manifest.json`. The contents of this manifest are defined by a second BDBag RO profile, described in §IV.

To enable the computation of an overall checksum for a BDBag for use in conjunction with Minid generation, the BDBag BagIt profile requires that each BDBag be serialized in ZIP, TAR+GZIP, or plain TAR format. This overall checksum is required so that a Minid can be generated for a bag with the assurance that the given bag data is what the creator intended.

To provide redundancy and flexibility in checksums, the BDBag BagIt profile requires that both MD5 and SHA256 checksum and checksum tag manifests be present, and requires that for any given file in the bag's payload directory there must be an MD5 and/or SHA256 checksum. (While MD5 is more widely used in implementations, it is less robust than SHA256 and collisions are possible.)

C. BDBags and Policy

One obvious benefit of the `fetch.txt` mechanism supports the exchange of a big dataset without copying large amounts of data. A second benefit is that it allows the definition of data collections for which specific individuals may not have immediate permission to access, as for example in biomedical applications in which certain access to data elements may be restricted by data use agreements.

In some situations, the data may not be accessible via standard URL access protocols. In this situation, the `fetch.txt` mechanism can be used to identify where to find the data, but not how to get it. For this purpose, we use a `tag` scheme URL [6]. Presented with a fetch file with tag URLs, the user can use an out of band mechanism to retrieve the data, position it in the payload directory, and then validate that the correct information is in the bag.

IV. Research Objects: Describing a BDBag

The BagIt specification and the BDBag BagIt profile provide a framework for enumerating the files that make up a potentially large and distributed dataset, in a way that allows a recipient of a BDBag to validate its contents. We also want to allow users to associate descriptive metadata with those files, and for that purpose we turn to the Research Object (RO) [7] framework.

The RO framework was developed to provide a model and mechanism for aggregating a set of digital resources that are viewed as constituting a *unit of research*: for example, the datasets, analysis scripts, and derived results associated with a research paper. To this end, an RO aggregates (a) these resources, (b) associated attribution and provenance information, and (c) structured and unstructured annotations describing the individual resources, their relations, and associations with external identifiers. These digital components can then be bundled into off-the-shelf containers as supported by platforms such as ZIP, BagIt, and Docker. ROs are designed to promote reproducibility [8] by enabling portability and structured rich semantic descriptions.

We adapt the RO model for BDBag by defining a BDBag RO profile [9], which builds on the RO Bundle specification [10], but using BagIt-compatible paths: e.g., the `data` directory contains the dataset (after fetching), while the `metadata` directory contains annotations and the RO `manifest.json` in JSON-LD format.

The manifest file lists the name, media type, and semantic type of each bag resource. It can also provide per-resource attribution, provenance, and annotations. For attribution, it uses ORCID to identify people. For provenance, it uses W3C PROV-O to describe the structured history trace of generated files. Annotations are provided in separate files, which may be unstructured (e.g., text file), semi-structured (e.g., tabular spreadsheet), or structured (e.g., linked data using vocabularies and ontologies).

Listing 1 presents an example of a structured annotation file represented in JSON-LD. This file describes the `numbers.csv` file stored in the BDBag payload using the Dublin Core ontology. The annotation file is stored in the `metadata` directory of the BDBag and does not need to carry the same name as the reference file (the name of the referenced file is encoded in the `@id` attribute). In this case, the annotation file describes the type, title and description of the `numbers.csv` file. Other types of annotation files, including those that represent provenance information, can also be added to the `metadata` directory.

Our adoption of RO notation also permits the representation of more complex inter-element relationships than the basic hierarchical naming. supported by the BagIt specification. For backward compatibility with existing tools, hierarchical naming gives us a simple way of organizing things that is consistent with functionality provided by file systems and existing ZIP management tools. However, for more sophisticated use, we can have more complicated (i.e., graph) relationships that are expressed using JSON-LD in the standard metadata that we include for RO descriptions.

Listing 1: RO annotations in JSON-LD (abbreviated)

```
{
 "@context": {
  "@vocab": "http://purl.org/dc/terms/",
  "dcmi": "http://purl.org/dc/dcmitype/Dataset"
 },
 "@id": "../../data/numbers.csv",
 "@type": "dcmi:Dataset",
 "title": "CSV files of beverage consumption",
 "description": "A CSV file listing the number of
     cups consumed per person."
}
```

V. Minimal Identifiers: Fixity and Simplicity

Having created a bag, we face the problem of how to share it with data consumers. Even a partial bag may be too large to conveniently copy via methods such as email. We could place the BDBag on some shared storage service and share a URL, but such unregimented user-assigned names can easily become unmanageable. We can encourage users to follow naming conventions, but without support to enforce uniqueness, immutability, availability, and resolvability, errors easily occur. Moreover, complications are exacerbated when the data themselves are shared or modified, as there is no method to link data contents with a name. We require a robust and convenient naming scheme that can easily assign unique identifiers to *any* data. With such names, researchers can uniquely reference and locate data, and share and exchange names (rather than the entire contents of a dataset) while being able to ensure that the contents (or version) of a dataset is unchanged.

Referring to BDBags via a *permanent identifier* rather than via an arbitrary URL can go a long way toward addressing these problems. A permanent identifier is a layer of indirection which provides a unique stable reference to data which may be located in multiple locations and to which we can bind specific descriptive metadata elements such as the author and creation date. Typically, the use of identifiers, such as a DOI for DataCite, is limited to significant publication events, as creating the identifier requires stepping outside of ones normal data analysis workflow. We anticipate that BDBags will be created at many points during a big data analysis process, and thus we want to minimize cost and overhead associated with creating an identifier. In summary, we would like to be able to provide names for digital objects that are *unambiguous* (given an object's name, I can be sure that its content is exactly that provided by its creator); *actionable* (given a name, I can discover essential characteristics of the associated object and potentially access

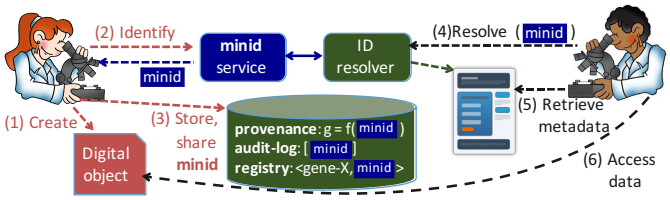

Figure 2: A Minid use case. See text for details

its contents); *disposable* (names are so easy and cheap to create that they can be created without thinking); *resolvable* (information about the name can always be obtained even if the data that it references is temporarily or permanently unreachable); and *persistent* (once minted, the name will persist for eternity and the attributes associated with that name, such as its creator, cannot be changed).

To support these needs we define a *minimal viable identifier* (Minid). As the name suggests, Minids are lightweight identifiers that can be easily created, resolved, and used. Minids are built upon Archival Resource Keys (ARKs) and take the form `ark:/57799/[suffix]`, where the shoulder (57799) is used for all Minids, and the suffix is a unique sequence of characters. Minids can be resolved using independent resolution services (e.g., name to thing). Minids resolve to a landing page that is designed to be always available, even if the data is not. The landing page includes a small number of metadata and references to locations of the referenced data. Rather than prescribe many required metadata (as in many publication systems), Minids require only *what* the data is, *when* it was created, and *who* created it. In order to ensure fixity, we also provide a cryptographic hash (checksum) of the contents of the object.

Figure 2 illustrates the use of Minids. A researcher images a biological sample and immediately associates a Minid with each image file, including the file's checksum and location. Using the Minid, she records notes in her electronic lab notebook that describe observations on each image. Having observed unusual results, the researcher sends the relevant Minids to a collaborator, who can then resolve them, using the name-to-thing (`https://n2t.net`) resolver. The collaborator can then download the files referenced by the Minid, verify that the files have not been changed, and refer to them in subsequent work and publications.

A. The Minid Service

The Minid architecture is centered around a hosted Minid service. The service provides a web-accessible location to create, manage, and resolve Minids. It offers a web interface for viewing landing pages, a REST API for programmatically creating and retrieving Minids, and a lightweight Python command line client for integrating Minids in researchers' working environments.

Minids are built upon ARKs, a general purpose identifier that support arbitrary information objects. Minids extend ARKs by defining an interface for creating, managing, and accessing them; a metadata model for describing data; and a lifecycle model that includes support for marking Minids as obsolete and no longer supported. The Minid service uses the EZID identifier service for minting ARKs.

We use a DataCite-based metadata schema to describe pertinent attributes of the Minid: *Creator* (the creator of the Minid, identified by name and/or ORCID); *Created* (the date of creation); *Title* (a text title of the Minid); *Locations* (URI locations for accessing the referenced data); and *Checksum* (a checksum for the referenced data).

The Minid service implements a lightweight authentication model to make the service easy to use and reduce technical burden on users. Users first register with the service by specifying their name, email address, and optional ORCID. The email address serves as a unique identity for a user. Users are sent a random text code that they can use in combination with their email address to use the service.

Minids implement a simple state model whereby a Minid can either be active or *tombstoned*. A tombstoned Minid represents data that are no longer available. A Minid may also be obsoleted by another Minid.

When resolving a Minid via a resolution service (e.g., name to thing), users are redirected to a web-based landing page: see Figure 3. The landing page presents the complete metadata record for the Minid and links to one or more locations where the referenced data can be obtained. It also includes a QR code that encodes the Minid and can be resolved using a QR code reader. The `GET` methods for the landing page support HTTP content negotiation and results may be returned in human-readable (HTML) or machine-readable (JSON) form.

B. Minids and BDBags

Minids are designed to identify any data, thus they can be applied to BDBags or used within BDBags to reference external data. The BDBag specification supports a variety of protocols for defining remote data elements to be included in the BDBag (via the `fetch.txt` file). When Minids are used as an identifier for an element, the BDBag tooling uses a resolver to retrieve the Minid's metadata and iteratively download the data from the specified locations.

It is natural to use a Minid to identify a BDBag for purposes of sharing, publication, and analysis. However, the varying states of completeness supported by BDBags are problematic for the checksum-based Minid architecture. For example, a fully complete BDBag with all data present in its `data` directory is semantically equivalent to an empty BDBag with its remote dependencies specified in the `fetch.txt` file, but these BDBags will have different checksums. Thus, Minids include a secondary *equivalence* hash so that data can be compared even if the physical bits in different bags are not identical. In the case of BDBags, an equivalence hash is constructed by taking a hash over

Figure 3: A Minid landing page for a BDBag generated by the ENCODE tool described in §VII-A

the bag-info, manifest, and tag-manifest files. Thus, the hash encodes the bag metadata, the full payload of the bag defined via the manifest, and the complete list of metadata files and checksums included in the bag.

VI. Tools

To ease user interactions with BDBags and Minids we have developed client tooling and libraries that support direct usage within researchers' workflows and integration with external applications. Here we briefly describe these tools.

A. The `bdbag` Utility

This Python program combines code forks of the BagIt-Python bag creation utility and the BagIt-Profiles-Validator utility into a single software package that greatly simplifies common tasks associated with creating, consuming, and maintaining bags. The utility provides an Application Programming Interface (API) that developers can call directly from their own Python programs, and an end-user command line interface (CLI) built on the API.

Other `bdbag` features are unique to the concept of a BDBag. It supports `fetch.txt`-based file retrieval from HTTP and Globus endpoints and resolution of Minids. Thus, a BDBag creator can fully leverage the Fixity of the data referenced by a Minid when including that data as part of a bag payload. We list other `bdbag` features in the following.

Update-in-place for existing bags. Many BagIt tools, including the Library of Congress reference implementation, BagIt-Python, do not provide a convenient mechanism for adding, removing, or changing a bag's content. The `bdbag` utility's smart update function automatically detects if a BDBag update would require regenerating checksums in all manifests (a potentially lengthy process with large files) or only tag file manifests.

Automatic archiving, extraction, and validation. Most bag tools require that bag archiving, extraction, and validation each be performed as separate steps. Built-in support for these tasks mean that common task combinations such as bag creation and archiving or bag extraction and validation can be performed as a single command. For example, the consumer of a BDBag can validate its profile, extract its contents (from a ZIP, TAR, or TGZ archive), resolve any remote file references, and validate the assembled BDBag all in a single command invocation.

Automatic generation of remote file manifest entries and `fetch.txt` via configuration file. Most bag tools have only weak programmatic support for the generation of remote file references. Generally, the bag creator must manually enter remote files in a relevant manifest while also creating a `fetch.txt` file and ensuring that the remote files listed in each manifest have corresponding lines in `fetch.txt`. During a bag update, if any remote files are added or removed the user must manually update both manifests and `fetch.txt` with the changes, which can be error-prone. The `bdbag` utility allows the BDBag creator to specify all required information in a single JSON-formatted configuration file and maintains coherency of all bag manifests and remote references in `fetch.txt` on behalf of the user.

Automatic file retrieval based on a bag's `fetch.txt` file, with multiple protocol support. Few BagIt tools support the fetching of files listed in a bag's `fetch.txt`. The `bdbag` utility provides an extensible mechanism for handling file fetching using multiple protocols.

Creation of bags with BDBag BagIt/RO profile compatibility. The `bdbag` utility supports the use of RO manifests, and provides an API for the creation of RO metadata and a bag profile that specifies the inclusion of such metadata.

The `bdbag` CLI and API are designed for ease-of-use and intended to operate on either BDBag directories or single file BDBag archives. It is context-driven, meaning that different functions are available based on the input BDBag path. For example, if the input path argument represents a path to a directory that is not already a BDBag, the software will create a BDBag around the contents of that directory using what is referred to as the *bag-in-place* mechanism. Bag-in-place simply moves all files present in the input directory to a temporary directory; creates the BDBag payload directory (i.e., `./data`); moves the original files from the temporary directory into the newly created payload directory; and finally generates all payload checksums and bag manifests and places them in the root of the original input directory. A user may optionally specify arguments that results in the automatic serialization (archiving) of the bag to one of the supported formats immediately following bag-in-place.

If the input path argument represents a path to a directory that is already a BDBag, a variety of functions are available.

A user may update an existing BDBag (e.g., add/remove/-modify payload files or bag metadata); resolve (download) remote file references listed in `fetch.txt`; validate the structure, integrity, and completeness of the BDBag; validate the BDBag's conformance to its profile (if any); or archive the BDBag to a supported format. Lastly, if the input path represents a serialized file representation of a BDBag, the software will by default extract the BDBag contents. However, it can also be used to validate the bag and/or bag profile, in which case the bag is extracted to a temporary directory that is removed after validation.

B. The `minid` Utility

The `minid` CLI and API enable integrated access to the Minid service from the command line or within an application. Once the command line client is installed, users can configure it by creating a *minid.cfg* file with their email address and code obtained from the registration workflow. The CLI enables creation of a Minid, retrieval of a Minid via checksum or Minid, or updating the state or location of a Minid, all directly from the command line.

When creating a Minid the CLI allows users to specify Minid metadata (e.g., title, location, and checksum) via command line arguments. Alternatively, the CLI can calculate a checksum and derive a location from locally accessible files. For example, the following call will create a Minid for `file.txt` with title "Foo" and with the creator defined in the configuration file, plus the file's checksum.

```
minid --register --title 'Foo' /path/file.txt
```

The CLI also supports the resolution of Minids. Given a Minid, a user can look up the metadata (including location) for the data that it references. Alternatively, users can look up a Minid via a file's checksum or equivalence hash via indexes maintained by the service. This is valuable as it allows users to discover Minids that might already be associated with a given file. The `minid` CLI is a Python application that can be installed on various operating systems.

C. Localizing Bag Contents using Globus

While BDBags make it easy to create a big dataset description and Minids enable unambiguous reference to large datasets, in many use cases it will be necessary to consolidate all of the data into a single location for subsequent analysis, processing, or archival. Standard BDBag tooling provides basic functions for consolidating the contents of a bag, however the default mechanism (HTTP) is unreliable and performs poorly without specialized methods to manage the download process. To address the needs of high performance and reliable transfer of large amounts of data both BDBag and Minid tooling use Globus [11].

We have integrated support for Globus in our Minid and BDBag tools to support high-speed access to big data. To support references to Globus accessible data we have adopted a Globus URI representation for inclusion in the location target of Minids or in the `fetch.txt` files of a BDBag. For example, the URI `globus://6a84efa0-4a94-11e6-8233-22000b97daec/path/file.txt` refers to the file `file.txt` on Globus endpoint `6a84efa0-4a94-11e6-8233-22000b97daec`. Minid and BDBag tooling is able to parse these references, identify that they refer to data accessible via Globus, and then use the Globus REST API to asynchronously transfer the data to the user-specified location (either locally or to another Globus-accessible location). For example, when Globus URIs are included in a BDBag `fetch.txt`, the BDBag tooling, when resolving the fetch file, will perform Globus transfers to download each file.

VII. Case Studies

We use three case studies to illustrate how our methods and tools are being used.

A. ENCODE Case Study

The ENCODE (Encyclopedia of DNA Elements) Consortium [12] is an international collaboration of research groups funded by the National Human Genome Research Institute (NHGRI). Its goal is to build a comprehensive parts list of functional elements in the human genome. The catalog includes genes (protein-coding and non-protein coding), transcribed regions, and regulatory elements, as well as information about the tissues, cell types and conditions where they are found to be active.

The ENCODE web portal allows researchers to perform queries using various parameters such as assay, biosample, and genomic annotations. A typical researcher workflow involves searching for data on the ENCODE portal, downloading relevant datasets individually, keeping track of all data that are downloaded, running various analyses on the data, creating results, and eventually publishing conclusions.

While online access to ENCODE data is a great boon to research, subsequent steps can become cumbersome. Each data file URL returned by a query must be downloaded individually, and data comes without associated metadata or context. Researchers must manually save queries if they wish to record data provenance. The provenance of intermediate datasets and analysis results is often lost unless the researcher diligently captures them. There is no way for a researcher to validate that a copy of an ENCODE dataset has not been corrupted, other than to download the data again.

We describe in the following how our tools can be used to realize the end-to-end scenario shown in Figure 4.

1) ENCODE Data Access: We used BDBag, Minid, and Globus tools to create a simple portal that allows the researcher to access the results of an ENCODE query, plus associated metadata and checksums, as a BDBag. Figure 5 shows this portal in action. The researcher enters an ENCODE query in REST format or uploads an ENCODE metadata file that describes a collection of datasets. The

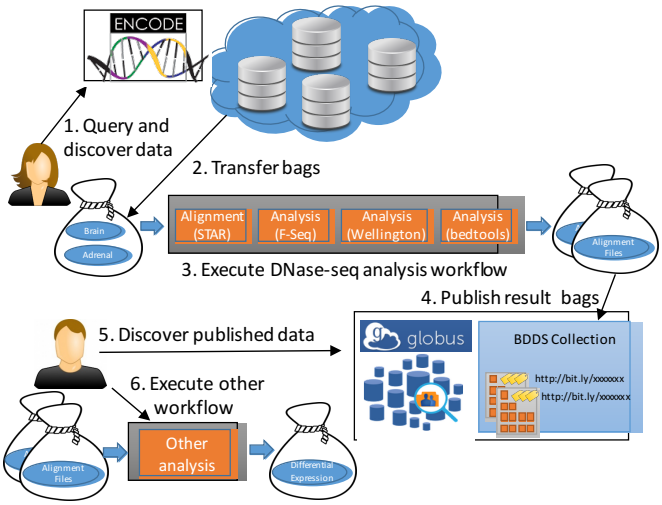

Figure 4: An end-to-end pipeline with ENCODE data: creating a BDBag, running pipeline, publishing results, reusing results

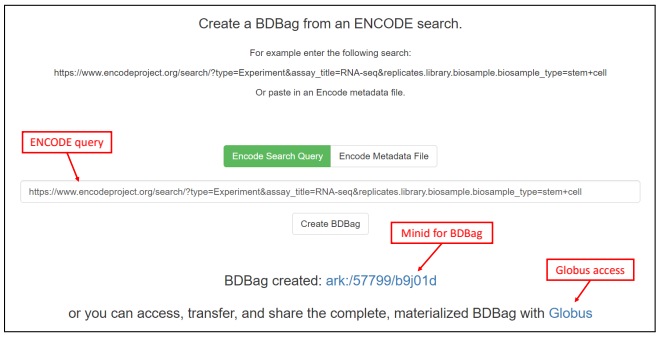

Figure 5: An ENCODE portal. The user has entered an ENCODE query and clicked "Create BDBag." The portal provides a Minid for the BDBag and a Globus link for reliable, high-speed access

query shown in Figure 5 identifies data resulting from RNA-Seq experiments on stem cells. (As of August 2016, the result comprises 13 datasets with 144 FastQ, BAM, .bigWig, and other files, totaling 655 GB of the 193 TB in ENCODE.)

The researcher can then select the "Create BDBag" button to trigger the creation of a 100 KB BDBag, stored on Amazon S3 storage, that encapsulates references to the files in question, metadata associated with those files, and the checksums required to validate the files and metadata.

As shown in Figure 5, the user is provided with a Minid for the BDBag, `ark:/57799/b9j01d`. This identifier leads to a landing page similar to that shown in Figure 3, from which the user can download the BDBag. The BDBag is stored in Amazon S3 and can therefore be permanently referenced for purposes of sharing, reproducibility, or validation. The BDBag materialization tool can then be used to fetch each file from the URL provided in the `fetch.txt` file.

Given the Minid for a bag, a user may copy the bag to a desired storage location and use the BDBag tools to retrieve remote files from the ENCODE repository. The default practice of making client-driven HTTP requests can be time consuming and is subject to failures. For example, we found that instantiating the 655 GB test bag of remote ENCODE data via HTTP took 36 hours and two restarts. The BDBag tools provides an effective alternative: if the remote data is located on a Globus endpoint, reliable and high performance Globus transfer methods can be used that improve performance and automatically restart interrupted transfers. Thus, for example, we were able to instantiate the same BDBag from Petrel, a storage repository at Argonne National Laboratory, to a computer at the University of Chicago Research Computing Center in 8.5 minutes, and from Amazon S3 to the same computer in 14 minutes. If BDBag elements are referenced using Minids, the BDBag tools have the option of preferring Globus URLs if they are provided. The checksumming of bag contents ensures that we have the correct data, regardless of source.

2) Analyzing ENCODE Data and Publishing Results: Having enabled direct, programmatic access to ENCODE query results plus associated metadata, we next want to automate the analysis of such results. To this end, we use the Galaxy [13]-based Globus Genomics [14] extended to resolve a BDBag via a Minid, accept BDBags as input, retrieve the referenced datasets, and run analytical pipelines at scale on cloud resources: Steps 2 and 3 in Figure 4.

The analysis is performed on data from a molecular biology technique called DNase I hypersensitive sites sequencing, or DNase-seq [15]. Such experiments generate a large volume of data that must be processed by downstream bioinformatics analysis. Currently available ENCODE data for our analysis protocol comprise 3–18 GB of compressed sequence. Depending on input sequence size, analysis can require 3–12 CPU hours per sample on a AWS node with 32 CPU cores. The final output file is relatively small compared to the input, but intermediate data is about 10 times the input size. The resulting output for each individual patient sample is encapsulated in a BDBag—containing a collection of candidate DNA footprints. All individual samples from the same cell line are merged and then filtered by intersecting against a database for known transcription factor binding sites in the reference genome. This analysis step takes one CPU hour and produces 50–100GB of output, depending on the cell type. The final BDBags are assigned a Minid. We also publish these BDBags into the BDDS Publication service (Step 4) [16], so that other researchers can discover and access the results for other analyses (Steps 5 and 6).

B. PeptideAtlas Case Study

Another resource that uses BDBags and Minids for disseminating data is the PeptideAtlas [17] proteomics data repository. PeptideAtlas collects proteomics tandem mass spectrometry (MS/MS) datasets from laboratories around the world, either by direct submission or via the ProteomeXchange Consortium [18], and reprocesses them with the ad-

vanced software tools of the Trans-Proteomic Pipeline [19], using the most recent reference proteome. All data products generated by PeptideAtlas are made publicly available.

Many reference proteomes and other protein lists can be used to process human proteomics MS/MS datasets, and it can be difficult to determine which is most appropriate for a given analysis. Thus, PeptideAtlas disseminates a Tiered Human Integrated Search Proteome (THISP) set, refreshed on the first of every month, at the PeptideAtlas web site via BDBags and Minids (`http://www.peptideatlas.org/thisp`) [20]. These monthly releases comprise a set of files of different levels of complexity, with and without decoy sequences for error calibration. Each release is bundled into two separate BDBags with associated Minids using the tools described herein; the first bundle is the primary proteome files, and the second is all individual components that may be used for custom applications.

In addition, PeptideAtlas makes all deposited and reprocessed datasets available to the community via its raw data repository. For each dataset, this repository provides the raw instrument files, the raw data converted to the Proteomics Standards Initiative's mzML [21] open standard format, the reprocessing results, and the metadata describing the experiment. The datasets have previously been made available as individual files on the FTP server, but are now bundled into BDBags with Minids to facilitate their dissemination to the community.

C. PheWAS Case Study

Phenome Wide Association Studies (PheWAS) are a big data analytics tool in which biomarkers are identified by analyzing many phenotypes from many subjects with a shared genetic trait, such as a common single nucleotide polymorphism. In one application, we generated thousands of phenotypes from neuroimaging data segmenting brain images against many different brain atlases, and then for each segmentation we computed many metrics, such as volume, density, and curvature. The BDBag and Minid tools are proving invaluable as we track, manage, and process the tens of thousands of files produced by PheWAS analysis.

As noted above, BDBags allow us to assemble and communicate data that are constrained by data use agreements. One initial dataset comprises 1028 files totaling 228 GB. A data use agreement prohibits us from copying the data from user to user: each user must go to the source repository to download the data. However, we can create a 64 KB partial bag in which every protected file is listed in the `fetch.txt` file and referenced using a `tag` scheme URL that includes the accession number of the file. A user receiving this bag can then formulate a download request from the repository by examining the contents of the manifest and fetch file. Once retrieved data is placed into the `data` directory for the bag, the BDBag tools can validate the contents to ensure that the right data has been obtained.

VIII. Related work

Our approach is differentiated from other efforts by its focus on developing an *integrated* suite of capabilities to manage, describe, and name *large* and potentially *distributed* data. We review efforts with similar goals and functionality to each of the core components of our approach.

The problem of describing the layout of heterogeneous datasets is addressed by formats such as the Hierarchical Data Format (HDF) [22], which defines a self-describing structure that allows many objects to be stored in a single file, and the DFDL [23] and XDTM [24] specifications, which use XML to describe the layout of data within a single file and across multiple files, respectively. Such formats are often used as a general base from which specialized formats, such as NetCDF, can be represented using various conventions. In general, such formats are complex, and require sophisticated tooling to access and consume data.

Many metadata representations provide for the description of scientific metadata. These models are typically developed by publication systems, aligned with different research communities, associated with execution models, or used for specific data types. Thus, they focus on individual aspects of a dataset. In a more general sense, efforts such as Open Archive Initiative's Object Reuse and Exchange (OAI-ORE) [25] provide a common model for standardizing representation of complex multi-object datasets and associated metadata. However, OAI-ORE does not define sufficiently detailed models for describing the relationship and contribution of objects to one another or to resources outside of the package.

A growing number of systems support the creation of persistent identifiers for digital citation. General systems such as DOIs, ARKs, and Handles [26] can be minted and associated with arbitrary digital objects. Each, however, comes with its own usage policies and management infrastructure. The University of California's EZID service provides a common API and management service for creating ARKs and DOIs. DataCite [27] supports DOI creation and provides a registry for discovering data based on registered metadata. Commercial services such as Bitly and TinyURL provide for the creation of short identifiers in place of long URLs. However these services do not support multiple locations, checksum integration, or updates to metadata. Identifiers.org and the MIRIAM Registry aim to capture cross-references via standard URIs [28], [29].

IX. Conclusions

The complexity of managing heterogeneous big datasets is often ignored but has significant consequences in terms of overheads, inefficiencies, and errors. We have introduced simple yet powerful tools that can be readily integrated into the daily workflow of scientists and into a wide range of analytic programs and services. Working with domain scientists, we have applied these tools in several applications.

Table I: How elements of our solution address requirements

	BDBag	RO	Minid	Globus	Tools
Enumeration	●	○	○	○	○
Fixity	●	○	●	●	○
Description	○	●	○	○	○
Identification	○	○	●	○	○
Distribution	●	●	○	●	○
Simplicity	○	○	●	●	●

The tools have been enthusiastically embraced by users. We summarize in Table I how the various elements of our solution address the six requirements of §II. Tools are available at `http://bd2k.ini.usc.edu/tools/`.

Acknowledgments

This work was supported in part by NIH contract 1U54EB020406-01, Big Data for Discovery Science Center [1]; DOE contract DE-AC02-06CH11357; and EC H2020-EINFRA-5-2015 grant 675728. We thank Kristi Clark, Segun Jung, and Alex Rodriguez for work on applications and Galaxy integration.

References

[1] A. W. Toga *et al.*, "Big biomedical data as the key resource for discovery science," *Journal of the American Medical Informatics Association*, vol. 22, no. 6, pp. 1126–31, 2015.

[2] N. A. Vasilevsky *et al.*, "On the reproducibility of science: Unique identification of research resources in the biomedical literature," *PeerJ*, vol. 1, p. e148, 2013.

[3] M. D. Wilkinson *et al.*, "The FAIR guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, p. 160018, mar 2016.

[4] J. Kunze *et al.*, "The BagIt file packaging format (V0.97)," Internet Engineering Task Force, Internet Draft (work in progress), draft-kunze-bagit-11.txt, Tech. Rep., 2015.

[5] K. Tabata *et al.*, "A collaboration model between archival systems to enhance the reliability of preservation by an enclose-and-deposit method," in *5th International Web Archiving Workshop*, Vienna, Austria, 2005.

[6] T. Kindberg and S. Hawke, "The 'tag' URI Scheme," RFC 4151 (Informational), Internet Engineering Task Force, Oct. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4151.txt

[7] S. Bechhofer *et al.*, "Research Objects: Towards exchange and reuse of digital knowledge," in *Workshop on The Future of the Web for Collaborative Science*, 2010, available from Nature Precedings http://dx.doi.org/10.1038/npre.2010.4626.1.

[8] A. González-Beltrán *et al.*, "From peer-reviewed to peer-reproduced in scholarly publishing: The complementary roles of data models and workflows in bioinformatics," *PLoS ONE*, vol. 10, no. 7, pp. 1–20, 07 2015.

[9] S. Soiland-Reyes, "Research Object BagIt archive," researchobject.org, https://w3id.org/ro/bagit. Visited July 1, 2016.

[10] S. Soiland-Reyes, M. Gamble, and R. Haines, "Research Object Bundle 1.0," Specification, researchobject.org, November 2014, https://w3id.org/bundle/2014-11-05/.

[11] I. Foster, "Globus Online: Accelerating and democratizing science through cloud-based services," *IEEE Internet Computing*, vol. 15, no. 3, pp. 70–73, 2011.

[12] ENCODE Project Consortium *et al.*, "The ENCODE (ENCyclopedia of DNA elements) project," *Science*, vol. 306, no. 5696, pp. 636–640, 2004.

[13] E. Afgan *et al.*, "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update," *Nucleic Acids Research*, vol. 44, no. W1, pp. W3–W10, 2016.

[14] R. K. Madduri *et al.*, "Experiences building Globus Genomics: A next-generation sequencing analysis service using Galaxy, Globus, and Amazon Web Services," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 13, pp. 2266–2279, 2014.

[15] L. Song and G. E. Crawford, "DNase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells," *Cold Spring Harbor Protocols*, vol. 2010, no. 2, pp. pdb–prot5384, 2010.

[16] K. Chard *et al.*, "Globus data publication as a service: Lowering barriers to reproducible science," in *11th International Conference on e-Science*. IEEE, 2015, pp. 401–410.

[17] E. W. Deutsch *et al.*, "State of the human proteome in 2014/2015 as viewed through PeptideAtlas: Enhancing accuracy and coverage through the AtlasProphet," *Journal of Proteome Research*, vol. 14, no. 9, pp. 3461–3473, 2015.

[18] J. A. Vizcaíno *et al.*, "ProteomeXchange provides globally coordinated proteomics data submission and dissemination," *Nature Biotechnology*, vol. 32, no. 3, pp. 223–226, 2014.

[19] E. W. Deutsch *et al.*, "Trans-Proteomic Pipeline, a standardized data processing pipeline for large-scale reproducible proteomics informatics," *PROTEOMICS-Clinical Applications*, vol. 9, no. 7-8, pp. 745–754, 2015.

[20] E. Deutsch *et al.*, "Tiered human integrated sequence search databases for shotgun proteomics," *Journal of Proteome Research*, 2016, submitted.

[21] L. Martens *et al.*, "mzML–a community standard for mass spectrometry data," *Molecular & Cellular Proteomics*, vol. 10, no. 1, pp. R110–000 133, 2011.

[22] The HDF Group. (1997–2016) Hierarchical Data Format, version 5. http://www.hdfgroup.org/HDF5/.

[23] M. J. Beckerle and S. M. Hanson, "Data Format Description Language (DFDL) v1.0 specification," Open Grid Forum, Tech. Rep. GFD-P-R.207, 2014.

[24] Y. Zhao *et al.*, "A notation and system for expressing and executing cleanly typed workflows on messy scientific data," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 37–43, 2005.

[25] C. Lagoze *et al.*, *Open Archives Initiative Object Reuse and Exchange*, http://www.openarchives.org/ore/1.0/vocabulary. Accessed August 1, 2016, Open Archives Initiative Std., October 2008.

[26] S. Sun, L. Lannom, and B. Boesch, "Handle system overview," Internet Engineering Task Force, Network Working Group, RFC 3650, November 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3650.txt

[27] J. Brase, "DataCite–A global registration agency for research data," in *4th International Conference on Cooperation and Promotion of Information Resources in Science and Technology*. IEEE, 2009, pp. 257–261.

[28] C. Laibe and N. Le Novère, "MIRIAM resources: tools to generate and resolve robust cross-references in systems biology," *BMC Systems Biology*, vol. 1, no. 1, p. 1, 2007.

[29] N. Juty, N. Le Novère, and C. Laibe, "Identifiers.org and MIRIAM Registry: Community resources to provide persistent identification," *Nucleic acids research*, vol. 40, no. D1, pp. D580–D586, 2012.