Distributed and Parallel Databases (2019) 37:273–295
https://doi.org/10.1007/s10619-018-7248-y

On-demand big data integration

A hybrid ETL approach for reproducible scientific research

Pradeeban Kathiravelu[1,2,3] · Ashish Sharma[1] · Helena Galhardas[2] · Peter Van Roy[3] · Luís Veiga[2]

Published online: 1 September 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract
Scientific research requires access, analysis, and sharing of data that is distributed across various heterogeneous data sources at the scale of the Internet. An eager extract, transform, and load (ETL) process constructs an integrated data repository as its first step, integrating and loading data in its entirety from the data sources. The bootstrapping of this process is not efficient for scientific research that requires access to data from very large and typically numerous distributed data sources. A lazy ETL process loads only the metadata, but still eagerly. Lazy ETL is faster in bootstrapping. However, queries on the integrated data repository of eager ETL perform faster, due to the availability of the entire data beforehand. In this paper, we propose a novel ETL approach for scientific data integration, as a hybrid of eager and lazy ETL approaches, and applied both to data as well as metadata. This way, hybrid ETL supports incremental integration and loading of metadata and data from the data sources. We incorporate a human-in-the-loop approach, to enhance the hybrid ETL, with selective data integration driven by the user queries and sharing of integrated data between users. We implement our hybrid ETL approach in a prototype platform, *Óbidos*, and evaluate it in the context of data sharing for medical research. *Óbidos* outperforms both the eager ETL and lazy ETL approaches, for scientific research data integration and sharing, through its selective loading of data and metadata, while storing the integrated data in a scalable integrated data repository.

Keywords Data integration · Scientific research · ETL (extract, transform, and load) · Big data

✉ Pradeeban Kathiravelu
pradeeban.kathiravelu@emory.edu

Extended author information available on the last page of the article

1 Introduction

Big data integration is crucial for numerous application domains, such as reproducible science [28], medical research [22], and transport planning [15], to enable data analysis and information retrieval. Scientific research often requires access to big data from various data sources, often geographically distributed [13]. Scientific data is typically heterogeneous, including binary and textual data, and stored in structured, semi-structured, or unstructured formats. Furthermore, data sources usually support distinct data access interfaces, ranging from database SQL queries to service-based application programming interfaces (APIs) [12]. Effectively and efficiently integrating such diversity and quantity of data is challenging.

To discover compelling scientific insights from data, it is often required to extract, transform, and load (ETL) it into an *integrated data repository* (e.g., a data warehouse [7]). This process is typically called ETL [32]. An ETL process makes data accessible through a uniform schema, by constructing an integrated data repository. Thus it supports fast and efficient querying of the scientific research data.

ETL efficiency: Traditionally, ETL has been an eager process, loading the entire content of the data sources into an integrated data repository as a first step. However, *eager ETL* is often unsuitable for handling scientific data. First, the bootstrapping process of eager data integration and loading takes too long. This time waste is unnecessary for scientific research [6] that often requires only a subset of data. Second, entirely integrating and loading the contents of data sources can be challenging due to the substantial resource demands. In fact, it requires high loading time and bandwidth. Furthermore, eager ETL also demands large storage due to the typical amount of data to integrate. Third, scientific data sources are often accessible only to authorized people. Loading the entire contents of data sources into an integrated data repository may enable to bypass the data authorization permissions established for data sources. Users would then be able to access data from the integrated data repository, thus increasing the probability of data access violation.

Lazy ETL [17] aims at mitigating the limitations of eager ETL, by integrating and loading the data only when necessary. Concretely, it avoids loading the entire contents of data sources into an integrated data repository as the initial step. A data source is composed of several data entries. For binary data, there is typically a piece of textual metadata (containing identifying information) attached to each data entry, in the file header. Metadata is often sufficient for the initial scientific research demands. As an illustrative example, consider the medical images stored in DICOM (Digital Imaging and Communications in Medicine) [27] standard format in various data sources such as the Cancer Imaging Archive (TCIA) [8]. The DICOM image file is in binary format. Often, there is textual metadata associated with each image. The DICOM metadata includes the image identification constituted by the series, study, and the identification of the patient the image belongs to. The metadata can be leveraged in the early stages of medical research, while DICOM image processing can be performed at a later phase, only for images selected as relevant (from the metadata). Thus, lazy ETL advocates for eagerly integrating and loading only the metadata, instead of the data entry itself (that is addressed lazily). Integrating and loading the metadata, in this case, is faster

than loading the entire data entry, due to the substantially smaller size of the metadata. Therefore, lazy ETL usually bootstraps faster than eager ETL.

Scientific experiments are often repeated several times by multiple researchers to confirm the accuracy of the outcomes. Therefore, frequent and repetitive queries are common. As a consequence, persistently storing the previously processed data entries into the integrated data repository would make recurring scientific research experiments faster. While eager ETL loads the data entirely into an integrated data repository, current lazy ETL approaches are not able to persistently store data required for previous queries. Therefore, recurring scientific queries execute slower under lazy ETL than under eager ETL. The gain obtained by faster data integration and loading in lazy ETL is lost when executing recurring queries because they cannot use stored results from previous queries.

Scalability: Scientific research often requires integrating large amounts of heterogeneous data from several web data sources [9]. Consequently, even an eager metadata-only ETL process (as prescribed by lazy ETL) can be challenging in scientific research, due to the distributed and heterogeneous nature of data sources. Moreover, metadata of some data sources tend to be as large as or larger than the data entries themselves. For example, Scality RING petascale object storage [29] consists of metadata up to 10 times larger than the data entries, supporting content-based searches through its metadata (designed for indexing). A typical lazy ETL process may fail to outperform an eager ETL process in bootstrapping in the presence of such data sources, due to the large size of metadata.

In practice, the researcher is often aware of the specific datasets that she needs and the characteristics of the data sources those datasets belong to. So, the researcher may be able to directly access the required data without accessing and querying the corresponding metadata. For example, consider a research study comparing the effects of an experimental medicine against those of placebo for variants of brain tumor. For this research study, the researcher only needs to load the imaging data of brain tumor from the data sources. Moreover, the researcher often possesses insights of the data such as the location of relevant image collections and the type of data access that is provided. Therefore, she can directly query the data sources and then load only specific subsets of the metadata, rather than eagerly loading the whole metadata.

Additionally, since the number of web data sources, as well as the amount of data and metadata, tend to increase, the storage requirements for the integrated data repository must be adaptable. In particular, a scalable storage is essential to accommodate data and metadata selectively accessed and incrementally integrated and loaded by the researcher. However, the current ETL approaches do not support such a selective ETL process into a scalable integrated data repository.

Interoperability and human intervention: Extracting and transforming data from web sources must consider various data storage and access interfaces. Data storage formats and access interfaces have been standardized in various research fields, to facilitate seamless access to the heterogeneous data sources. For example, Health Level Seven International (HL7) Fast Healthcare Interoperability Resources (FHIR) [14] is a standard for consistent data exchange between healthcare applications. Despite the popularity of these standards, a vast majority of data sources still fail to adhere to them. Thus, interoperability between heterogeneous data sources is still lacking [16]. Con-

sequently, data integration across various scientific web data sources is challenging, and typically not effective and efficient without human involvement.

Currently, in some domains, ETL is performed on-demand by a user [19]. The user is involved in the ETL process by incrementally integrating and loading subsets of data or metadata that are relevant to a given research question. The user is often aware of the details about data source access and data location. This expert knowledge could and should be incorporated into the ETL process. This type of user involvement is called *human-in-the-loop ETL*. It often consists of two parts. First, the user manually searches and downloads the datasets from the web data sources. Then, she integrates and stores the result in an integrated data repository. By narrowing down the search space to a smaller subset of relevant data sources, human-in-the-loop ETL shortens the data integration and loading time. However, existing ETL frameworks do not support the automatic incorporation of human in the process. Therefore, currently, human-in-the-loop ETL process remains a cumbersome manual and repetitive task.

Efficient scientific data sharing: Data used in a scientific research study often needs to be shared among researchers for collaboration and reproducibility purposes. However, this process is not efficient. First, sharing data by replicating its contents creates an excessive overhead on bandwidth, storage, and data maintenance. Therefore, data must be shared with minimal data replication. Second, researchers interested in the data resulting from an integration process may belong to one or many organizations. Repetition of the ETL process to obtain the same integrated data must be avoided, even when the collaboration extends beyond the organizational boundaries. A typical use case among the medical research scientists is to virtually integrate datasets from heterogeneous distributed data sources and share the results among the collaborators. Current approaches are inefficient in such data sharing facilitated by a data integration beyond the organizational boundaries. Therefore, the integrated data are often manually shared, in an approach oblivious to the ETL process. Such data sharing is inefficient and may lead to the existence and maintenance of duplicate data. A distributed ETL process to support sharing of the integrated data, with minimal repeated data loading and integration efforts with a minimal bandwidth overhead, is still lacking.

Motivation: Given the above premises, we aim at addressing the following research questions in this paper:

- (RQ_1) Can we increase the speed of the bootstrapping process in ETL by selectively accessing, integrating, and loading metadata?
- (RQ_2) Can we achieve faster execution time for repetitive scientific research queries by storing the previously integrated and loaded data in an integrated data repository?
- (RQ_3) Can we incorporate the human knowledge into an ETL framework to selectively and incrementally integrate and load only the relevant subsets of metadata or data, from web data sources?
- (RQ_4) Can the relevant subsets of data and metadata loaded by a research scientist for a specific experiment be shared efficiently for reproducibility purposes, thus minimizing data replication across peers from multiple organizations and avoiding the repetition of the ETL process?

Contributions: The goal of this paper is to answer the identified research questions, focusing on medical research as motivating real-life domain. The main contributions of this paper are:

(1) A novel hybrid ETL approach for accessing and integrating data and metadata from heterogeneous data sources, and loading the resulting data into a scalable integrated data repository. (RQ_1 and RQ_2)
(2) The incorporation of human knowledge into a hybrid ETL process to selectively integrate and load subsets of data and metadata on-demand. (RQ_3)
(3) A data sharing mechanism that enables to virtually share the relevant datasets efficiently through "pointers" to data, instead of repeatedly loading and replicating the actual data and metadata. (RQ_4)

We implemented *Óbidos*,[1] an on-demand big data integration platform for scientific research. We presented a preliminary version of *Óbidos* in our previous work [18]. In this paper, we elaborate in detail, how *Óbidos* supports hybrid ETL enhanced with human-in-the-loop for efficient data sharing. We deployed and performed an experimental evaluation of *Óbidos* for medical research data. In particular: (i) we compared *Óbidos* data loading and query execution times with eager and lazy ETL, and (ii) we evaluated the efficiency of *Óbidos* regarding the amount of data replication and bandwidth required in data sharing. The results obtained indicate that *Óbidos* performs better than or equal to both eager and lazy ETL approaches. We further observed that *Óbidos* data sharing feature avoids data replication and repeated ETL efforts.

Paper organization: The rest of this paper is structured as follows: Sect. 2 presents the solution architecture of *Óbidos*. Section 3 describes the implementation details of the *Óbidos* prototype. Section 4 presents the experimental evaluation that we conducted and the results obtained. Section 5 discusses the related work on data integration, data sharing platforms, and ETL approaches for scientific research. Finally, Sect. 6 concludes with a summary of the current status and future research directions.

2 *Óbidos*: an on-demand big data integration platform

The *Óbidos* platform is instantiated for each organization. Users from the organization can access, integrate, and load data into the integrated data repository of the corresponding *Óbidos* instance. Furthermore, they can share datasets stored in the integrated data repository with other users from the same or different organizations. Section 2.1 presents the *Óbidos* hybrid ETL approach and the underlying architecture. Section 2.2 explains how *Óbidos* incorporates human knowledge in the ETL process to selectively and incrementally integrate and load subsets of data and metadata. Section 2.3 details *Óbidos* efficient data sharing mechanism beyond organization boundaries to minimize data replication and repeated ETL efforts.

[1] *Óbidos* is a medieval fortified town that has been patronized by various Portuguese queens. It is known for its sweet wine, served in a chocolate cup.

2.1 Hybrid ETL process

Óbidos architecture: Figure 1 depicts the architecture of an *Óbidos* instance. From bottom to top, *Óbidos* consists of (i) a scalable **Integrated Data Repository**, (ii) a **Data Management Layer** with constructs for fast data integration and loading, and (iii) a **Query Rewriter** with constructs for efficient and unified access to the data in the integrated data repository and the data sources.

The **Integrated Data Repository** incrementally stores the data and metadata integrated and loaded by users. It consists of (i) structured and unstructured data (including binary data) as **integrated data** and (ii) the corresponding metadata as **integrated metadata**. Furthermore, the metadata stored in the integrated data repository needs to be indexed for efficient query execution over the binary data. We call this index of the integrated data repository, the **Metadata Index**. The Metadata Index functions as an internal index that is built over the integrated data and metadata in the *Óbidos* instance. *Óbidos* further stores the incomplete metadata entries, the metadata that is being loaded, as **virtual proxies**. The virtual proxies are stored as *future* or a placeholder for the complete metadata in the integrated data repository. The complete metadata will replace the virtual proxies once the entire metadata is loaded.

The **Data Management Layer** consists of data structures to manage the data in the integrated data repository and components to access, integrate, and load from the data sources. It stores its data structures in memory in a cluster of machines, aiming to offer fast access to the integrated data while not compromising fault-tolerance. A *virtual replica* is a pointer to a dataset from a distinct data source. A *replicaset* is an *Óbidos* data structure that is composed of several virtual replicas. Thus, each replicaset points to the distributed and diverse datasets relevant to a scientific research study. Furthermore, the replicasets are identified by timestamps. Therefore, the integrated

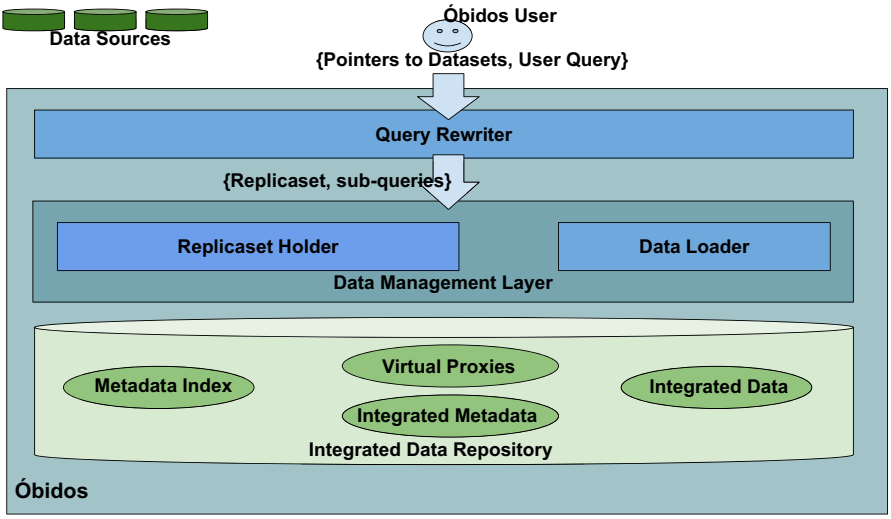


Fig. 1 *Óbidos* architecture

data repository can be periodically updated with the changes or updates to the datasets in the data sources pointed by the replicasets.

The **Replicaset Holder** is the core module of the Data Management Layer. It identifies each replicaset by a globally unique identifier known as replicasetID. The *Replicaset Holder* stores the replicasets in memory in a data structure that maps each item of integrated and loaded metadata into the corresponding replicasets. Thus, it indicates which of the datasets have already been loaded into the integrated data repository, either as integrated metadata and data or as virtual proxies. Moreover, it enables sharing the replicasets among users freely to make the datasets relevant to the scientific research available to other participants. Therefore, it serves as a component that prevents repetitive attempts to access, integrate, and load the same datasets. The **Data Loader** selectively loads metadata and data from the data sources. The location and the access mechanisms to the data sources are provided by the user and are stored in memory by the Data Loader.

Finally, the **Query Rewriter** enables uniform access to data sources as well as to the integrated data repository. It accepts as input a user query and pointers to the relevant datasets. Then, it converts the pointers to the datasets into replicasets. It also translates user queries into sub-queries that access either the data sources or the integrated repository. If the data required to answer the user query is not present in the integrated data repository, it invokes the Data Loader to integrate and load the datasets to answer the user query as well as the virtual proxies corresponding to the replicaset. **Óbidos** *incremental data integration and loading*: *Óbidos* accesses data and metadata from the data sources, and incrementally integrates and loads the results of the user queries into an integrated data repository. The integrated data repository persists previous query answers as well as the data and metadata integrated and loaded for answering previous queries. Therefore, queries can be regarded as virtual datasets that can be re-accessed or shared (akin to the materialized view in traditional RDBMS).

Óbidos enables to incrementally integrate and load metadata to mitigate the challenges in loading the metadata entirely or eagerly. When incrementally loading the metadata, *Óbidos* replaces the counterparts of metadata that has not been loaded yet with a virtual proxy. The use of virtual proxies minimizes the volume of metadata integrated and loaded. *Óbidos* stores the virtual proxies in the integrated data repository in addition to the integrated data and the corresponding metadata. If only a fraction of metadata is relevant for a search query, it is sufficient to load only that fraction. Therefore, *Óbidos* selectively loads metadata as virtual proxies. The virtual proxies are later replaced by the complete metadata as the metadata is accessed and integrated. Thus, virtual proxies refer to the metadata of a dataset larger than that is integrated and loaded to the integrated data repository.

Often a virtual replica may be present in the Replicaset Holder, without having the exact data for the user query. This usually means, previously at least one different user query has been executed on the same virtual replicas. Therefore, while the virtual proxies of the replicaset are present, the exact data for the user query may not be present in the integrated data repository. With time, as more and more data are selectively integrated and loaded, the integrated data repository will contain the necessary data for the subsequent scientific research queries.

2.2 Human-in-the-loop ETL process

Óbidos supports a human-in-the-loop ETL process. By 'human-in-the-loop', here we mean to incorporate the human knowledge that corresponds to the user-defined replicasets and queries to selectively access and integrate data from the data sources and incrementally loading the integrated data repository. A user identifies certain datasets as relevant to her scientific research, and these datasets are the ones against which the user query will be executed. She defines a replicaset by including pointers to these datasets as virtual replicas. The replicaset and a specific user query determine the data to be integrated and loaded by each selective data integration and loading process. This avoids the need to look for the desired data across data sources exhaustively.

The *Óbidos* selective load process is initiated every time a user issues a query. First, *Óbidos* iteratively checks for the existence of the data necessary to answer the query in the instance. It queries the Replicaset Holder for each of the virtual replicas and then executes the user query on the integrated data repository. If the data is not available in the instance, it is integrated and loaded from the data sources. The results of the user queries are persistently stored into the integrated data repository. Furthermore, rather than just querying and loading only the answers of the user query, *Óbidos* selectively loads the metadata pointed by the replicaset. This ensures that the integrated data repository can be incrementally loaded with data, rather than merely storing discrete, incoherent, or independent sets of data.

Figure 2 shows an *Óbidos* user defining a replicaset along with a user query to be executed on multiple data sources. The replicaset narrows down the search space from the entire data sources to specific datasets to answer the user query. She ensures with the knowledge of the data sources, the data required to answer her user query is part of the datasets pointed by her replicaset.

The data integrated and loaded into the integrated data repository of an *Óbidos* instance should be available to be accessed later for scientific research. For example, when a user receives a replicaset from another user from the same or another organization, she may access her organization's instance to check for already loaded data. Algorithm 1 illustrates how a user initiates the selective and incremental data integration and loading process of *Óbidos*.

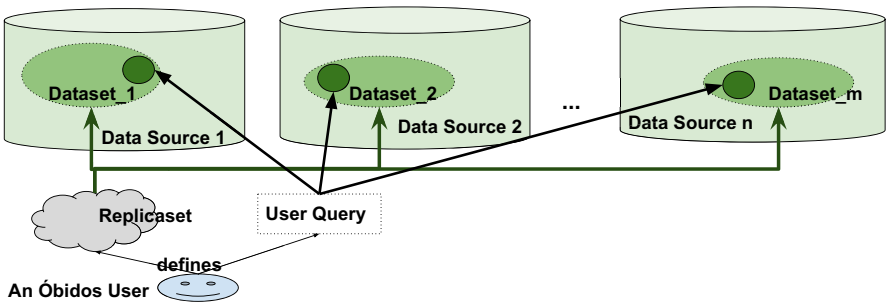


Fig. 2 Narrowing down the search space with user-defined replicasets

Algorithm 1 *Óbidos* Human-in-the-Loop Incremental ETL

```
1:  procedure SELECTIVELOAD(replicaset, userQuery)
2:      toLoad ← replicaset
3:      for all (virtualReplica in replicaset) do
4:          wasLoadedBefore ← replicasetHolder.get(virtualReplica)
5:          if (!(wasLoadedBefore) then
6:              loadData(virtualReplica, userQuery)
7:              replicasetHolder. put (virtualReplica)
8:              toLoad.delete(virtualReplica)
9:          end if
10:     end for
11:     if ((toLoad.size > 0) AND
                (integratedDataRepository.query(userQuery) == NULL)) then
12:         for all (virtualReplica in toLoad) do
13:             loadData(virtualReplica, userQuery)
14:         end for
15:     end if
16: end procedure
```

The algorithm starts by initializing a temporary variable *toLoad*, as a set, with the copy of the replicaset (line 2). *toLoad* tracks the virtual replicas belonging to the replicaset that have not yet been loaded from the data sources. Then, the algorithm proceeds to check the existence of the data pointed by each virtual replica in the instance (line 3). First, it queries the Replicaset Holder to check whether datasets pointed by the virtual replica have already been loaded by a previous query (line 4). If no dataset has yet been loaded for the virtual replica (line 5), the data relevant for the virtual replica and the user query is loaded from the data sources incrementally, invoking the *loadData* procedure (line 6). The Replicaset Holder matches the replicasets to the respective data and metadata integrated and loaded in the integrated data repository, by the selective load process. Therefore, in line 7, the virtual replica is added to the Replicaset Holder. Now since the dataset pointed by the virtual replica has already been loaded, the virtual replica is removed from *toLoad* (line 8).

The first loop (lines 3–10) checks whether the data, metadata, or virtual proxies relevant for one or more of the virtual replicas exist in the integrated data repository. It loads the data only when neither corresponding data and metadata nor virtual proxies are found for a given virtual replica. Therefore, a non-empty set of *toLoad* at the end of the loop indicates that at least a few virtual replicas were not loaded during this iteration. In that case, the algorithm proceeds to check whether the data and metadata necessary to answer the current user query are completely available in the integrated data repository (line 11). The user query will return a NULL if the complete metadata and data necessary to answer the query are not present in the integrated data repository. Consequently, the *loadData* procedure is executed for all the virtual replicas in the *toLoad* set (lines 12–14).

The loadData procedure: The loadData procedure is the core of the *Óbidos* human-in-the-loop incremental ETL approach. It accepts a replicaset and a user query as input arguments. First, the data sources are accessed, and the datasets identified by the replicasets are selectively loaded as virtual proxies, without loading the entire metadata. Then, the user query is executed against the data sources. The relevant metadata representing the results of a user query is integrated and loaded to the integrated data repository. If the user query also indicates access to the binary data, the respective binary data (usually a subset of data corresponding to the metadata already loaded by the query) is also loaded to the integrated data repository. The *loadData* procedure selectively loads the metadata corresponding to the replicaset as virtual proxies. If previously a different user query was issued with the same virtual replica, the virtual proxies corresponding to the virtual replica would be present while the exact data and metadata to answer the current user query would be absent in the integrated data repository.

2.3 Data sharing process

An *Óbidos* instance is deployed in each organization. Each *Óbidos* instance is used by: (i) users from the organization, and (ii) users from other organizations and external users who have limited access to the *Óbidos* instance. Users can share the datasets among them by sharing the replicasets or their respective replicasetIDs. Therefore, there is no need to replicate the actual data of the data sources nor the integrated data repository of an *Óbidos* instance.

Replicasets are small in size. However, they grow with the number of data sources and diversity of data. ReplicasetID is smaller in size compared to the replicaset and is of a fixed size. Therefore, they are shared by default. A user outside the organization can access the data already loaded in an *Óbidos* instance using the replicasetID. Moreover, users can share the replicasets with other organizations, without letting them access the data in their integrated data repository. The organizations can then integrate and load the datasets pointed by the replicaset, from the data sources. The relevant datasets pointed by the received replicaset can later be integrated and loaded by the remote users to their own *Óbidos* instance.

Figure 3 illustrates the process of data sharing between users *User_s1* and *User_r1* from two different organizations (called sender and receiver). The sender organization and the receiver organization can also represent the same organization if both users belong to the same organization. Datasets can be shared by as a replicaset or the respective replicasetID.

Algorithm 2 describes the data sharing procedure executed by the *Óbidos* instance of the receiver organization. It takes as input: a replicaset (or its replicasetID) received from another user, the identification of users that created/sent and received the replicaset, and an optional object known as *accessSender* (line 1). A null value for the *accessSender* object indicates that the shared datasets should be accessed from the data sources. A non-null value indicates that the datasets need to be accessed directly from the sender instance. The *accessSender* object consists of relevant access mechanisms such as the access key to the integrated data repository of the sender instance.

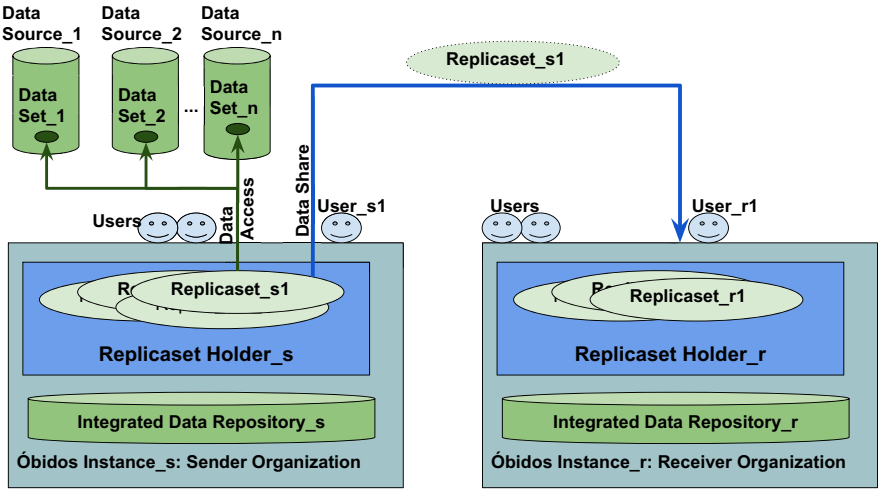


Fig. 3 Data sharing with *Óbidos*

Algorithm 2 Data Sharing via a Replicaset

```
1:  procedure SHAREREPLICASET(replicaset, sender, receiver, accessSender)
2:      if (replicaset.isURI())
3:          replicaset ← sender.get(replicaset) then
4:      end if
5:      if (accessSender != NULL)
6:          sender.access(replicaset) then
7:      else
8:          receiver.selectiveLoad(replicaset, NULL)
9:      end if
10: end procedure
```

If a replicasetID is received, the replicaset is retrieved from the sender instance first (lines 2–4). Since the replicaset was initially created by a user of the sender organization, the datasets or the virtual proxies pointed by the replicaset would be present in the sender organization. Therefore, if the *accessSender* is set to a non-null value (line 5), the datasets pointed by the replicaset are accessed directly from the sender instance, by the receiver organization (line 6). Otherwise, the *shareReplicaset* procedure selectively loads the datasets pointed by the replicaset into the receiver instance, from the data sources (line 8). As there is no user query defined in a shared replicaset, the *selectiveLoad* procedure is invoked with a null value in place of the user query.

3 Implementation

We built *Óbidos* with several composable data services, including data cleaning, loading, and sharing. *Óbidos* consists of data structures, APIs, and software components that enable chaining of these data services for its execution. Thus, *Óbidos* builds its hybrid ETL process with data sharing, effectively as a distributed ETL process beyond organizational boundaries.

3.1 Data structures

The Replicaset Holder stores the replicasets in a minimal tree-like data structure, to offer efficient search and indexing capabilities. Figure 4 illustrates the data structures of the Replicaset Holder and the data representation of *Óbidos*. The Replicaset Holder consists of a few instances of the *multi-map* data structure, where a set of items is stored as the value in the map, against a given key. As each user composes several replicasets, the **userMap** stores a list of replicasets against the identification of the users (userID) that created them.

Each entry in the list of values of the userMap represents a replicaset of a user. Each such entry in the userMap points to a **replicasetMap**, which includes the virtual replicas belonging to each replicaset, and whether the replicasets have already been integrated and loaded to the integrated data repository. A **replicasetID** is a globally unique random value generated by appending a random string generated via the Java's random UUID generator (using *UUID.randomUUID().getLeastSignificantBits()*) to

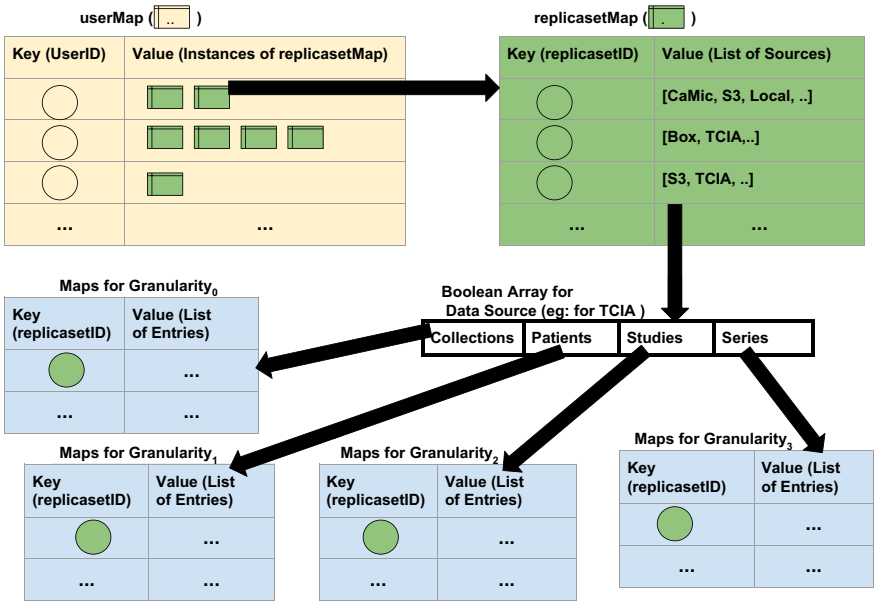


Fig. 4 Data structures of the Replicaset Holder

the URL of the *Óbidos* deployment. The *replicasetMap* employs the replicasetID as its key and the list of data source names contributing data to a replicaset as the value. Thus, each *replicasetMap* stores the relevant data sources for each of the replicaset.

n maps internally represent each data source belonging to a replicaset. In a hierarchical data storage format such as DICOM, each of the n maps represents one of the granularity levels in the data source. Such a format facilitates seamless integration of virtual proxies into the metadata. A boolean array A_b of length n is used to represent the replicaset in a bit-map like manner. Each element $A_b[i]$ of the array represents the existence of a non-null entry in the ith map of granularity. Thus, the boolean flags in A_b indicate the presence (or lack thereof) of the dataset in a particular granularity. If an entire level of granularity is included in the replicaset by the user, the relevant flag is set to true.

Replicasets include pointers to datasets from various data sources as virtual replicas. It leads to several maps: one map for each granularity of the data sources (for example, collections, patients, studies, and series for DICOM images). Each of these maps stores replicasetIDs as their keys and list of the entries as their values, in the given granularity (shown as $Granularity_0$, $Granularity_1$). Figure 4 represents an illustrative use case for a hierarchical data storage. It considers cancer images of DICOM format stored in data repositories such as TCIA, S3 buckets, directories in Box.com, and a local folder/file hierarchy. For these cancer images of DICOM format, $n = 4$. Thus, a map represents each of its four granularity levels—collections, patients, studies, and series, with an array of length 4 pointing to each of the four maps. The hierarchical data representation enables incremental loading and virtual proxies through its indexed data structure.

3.2 Service-based APIs

The *Óbidos* APIs are designed as CRUD (Create, Retrieve, Update, and Delete) functions on replicasets. Its functions are exposed as RESTful services, POST, GET, PUT, and DELETE. *Óbidos* offers a *data sharing* API to share scientific research datasets, by sharing the replicasets. Replicasets can also be shared outside *Óbidos*, through other communication media such as email. The data sharing method is typically one-to-one, meaning that a user shares data with another user in the same or different organization. However, it can also be listed for the public to be freely accessed.

The user accesses, queries, integrates, and loads the relevant data from the data sources by invoking the ***create*** *replicaset* procedure. This procedure creates a replicaset and initiates the selective data integration and loading process. When ***retrieve*** *replicaset* is invoked, the data corresponding to the given replicaset is retrieved from the integrated data repository. Furthermore, *Óbidos* checks for updates from the data sources pointed by the replicaset, if the data corresponding to the replicaset has already been integrated and loaded. Metadata of the replicaset is compared against that of the data sources for any corruption or local changes. The user deletes existing replicasets by invoking the ***delete*** *replicaset*. When a replicaset is deleted, the Replicaset Holder is updated immediately to avoid loading updates to the deleted replicasets. The user updates an existing replicaset to increase, decrease, or alter its scope, by invoking

the ***update*** *replicaset*. Thus, the update process may, in turn, invoke parts of create and delete processes, as new data may be loaded while existing parts of data may be removed.

The Replicaset Holder associates each dataset to a user, through its data structures such as the userMap. While each user has her own virtually isolated space in memory, the integrated data repository consists of a data storage shared among all the users of the organization. Hence, before deleting a data entry from the integrated data repository, the data should be confirmed to be an 'orphan' with no replicasets referring to them from any of the users. Deleting data from the integrated data repository is designed to be initiated by a background task, rather than its regular users. When the storage is abundantly available in a cluster, *Óbidos* advocates keeping orphan data in the integrated data repository rather than immediately initiating the cleanup process, and repeating it too frequently.

3.3 *Óbidos* software components

Óbidos architecture consisting of its data structures and interfaces is generic and can even be exploited for the integration of data from data sources other than the medical research data. We exploit several open source frameworks as major dependencies in our *Óbidos* prototype. Apache Hadoop Distributed File System (HDFS) [33] is used as the core of the integrated data repository, due to its scalability and support for storing unstructured and semi-structured, binary and textual data. *Óbidos* executes on a cluster of Infinispan [25] in-memory data grid. Consequently, the Data Management Layer stores its data structures in an Infinispan cluster. The metadata of the binary data in HDFS is stored in tables hosted in Apache Hive [31] metastore based on HDFS. The Hive tables consisting of the metadata are indexed with the Metadata Index for users to query and locate the data from the integrated data repository efficiently.

Apache Drill enables SQL queries on structured, semi-structured, and unstructured data. Therefore, the Query Rewriter unifies and accesses the storages seamlessly by leveraging Apache Drill [11]. Thus, *Óbidos* supports SQL queries on unstructured data stored in HDFS, through the Metadata Index stored in Hive. This approach allows efficient queries to the data, partially or wholly loaded into the integrated data repository. Thus, *Óbidos* provides unified and scalable access to the data in the integrated data repository and the data sources.

Oracle Java 1.8 is used as the programming language in developing *Óbidos*. Apache Velocity 1.7 [10] is leveraged to generate the application templates of the *Óbidos* web interface. Hadoop 2.7.2 stores the integrated data along with its corresponding metadata and virtual proxies, while the Metadata Index is stored in Hive 1.2.0. Hive-jdbc package writes the Metadata Index into the Hive metastore through its JDBC bindings to Hive. SparkJava 2.5 [30] compact Java-based web framework is leveraged to expose the *Óbidos* APIs as RESTful services. The APIs are managed and made available to the relevant users through API gateways. API Umbrella is deployed as the default API gateway. *Óbidos* incorporates authorization to its shared data from the integrated data repository through the use of API keys, leveraging the API gateway.

Thus, one can only access the data shared with them, and only with the API key that belongs to them.

Embedded Tomcat 7.0.34 is used to deploy *Óbidos* as a web application. Infinispan 8.2.2 is used as the In-Memory Data Grid where its distributed streams support distributed execution of the hybrid ETL processes across the *Óbidos* clustered deployment. The data structures of the Data Management Layer are represented by instances of the Infinispan Cache class, which is a Java implementation of distributed HashMap. Drill 1.7.0 is exploited for the SQL queries on the integrated data repository, with drill-jdbc offering JDBC API to interconnect with Drill from the Query Rewriter.

While the architecture and design are generic and can be extended for various data sources, our prototype implementation narrows down its focus to the medical research data, including textual and binary data that adhere to formats such as DICOM. We separate the implementation from the interface to ensure the reusability of the framework.

4 Evaluation

We benchmarked *Óbidos* against the implementations of eager ETL and lazy ETL approaches, using microbenchmarks derived from medical research queries on cancer imaging and clinical data.

Evaluation environment and benchmark analysis: An *Óbidos* prototype, implemented as described above, has been deployed to integrate medical data from various heterogeneous data sources including The Cancer Imaging Archive (TCIA) [8], DICOM imaging data hosted in Amazon S3 buckets, medical images accessed through caMicroscope [5], clinical and imaging data hosted in local data sources including relational and NoSQL databases, and file system with files and directories along with CSV files as metadata. The core data used in the evaluations are DICOM images. They are stored as collections of various volume as shown in Fig. 5. The data consists of large-scale binary images (in the scale of a few thousand GB, up to 10,000 GB) along with a smaller scale textual metadata (in the range of MBs).

Figure 6 illustrates the number of patients, studies, series, and images in each of the collection. Collections are sorted according to their total volume. Each collection

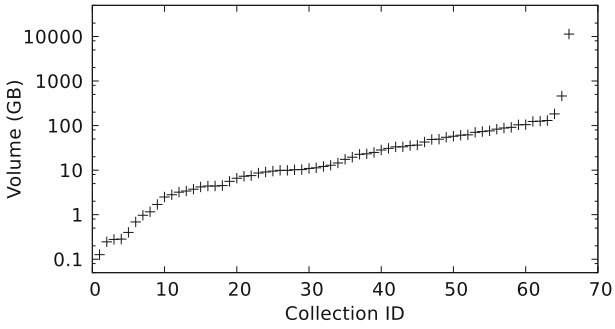


Fig. 5 Evaluated DICOM imaging collections (sorted by total volume)

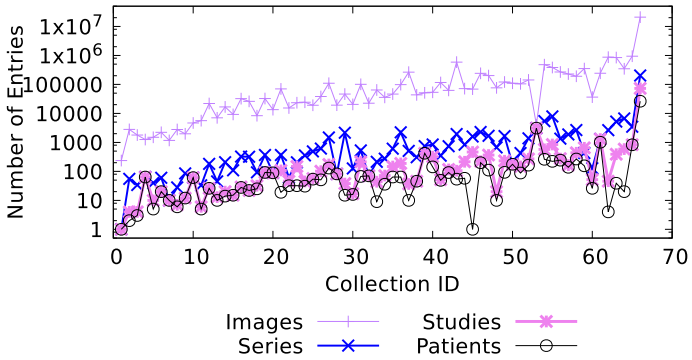


Fig. 6 Various entries in evaluated collections (sorted by total volume)

consists of multiple patients, each patient has one or more studies, each study has one or more series, each series has numerous images. We defined replicasets at these different levels of granularity. The varying pattern of Fig. 6, when compared against that of Fig. 5 shows that the total volume of a collection does not necessarily reflect the number of entries in it.

4.1 Performance of integrating and loading data

Óbidos was benchmarked for its performance and efficiency in integrating and loading the data. *Óbidos* integrates and loads data from the scientific research data sources spanning the globe. Therefore, the performance of loading the data will be influenced by the bandwidth. To avoid this influence, first, we replicated the data sources such as TCIA to data sources hosted on the local servers.

We integrated and loaded data from different total volumes of data sources for the same replicasets of the user. We measured the volume of the data sources by the total number of studies in them. Figure 7 shows the data load time of *Óbidos* against that of lazy ETL and eager ETL approaches. Since *Óbidos* selectively loads the metadata of only the data corresponding to the replicaset, the loading time remained constant independent of the increasing total volume of data in the data sources. However, since lazy ETL and eager ETL approaches query the entire data sources, the increase of volume leads to a larger time to integrate and load them. Eager ETL always took more time as it has to integrate and load the entire metadata and data. Since lazy ETL loads only the metadata eagerly, it loads faster than eager ETL.

Furthermore, for smaller volumes of data, eagerly loading the entire metadata can be faster than the selective loading by *Óbidos*, as *Óbidos* executes the query on the data source and loads the virtual proxies, creating and updating the constructs such as the Metadata Index and the Replicaset Holder. Therefore, *Óbidos* took longer for the data integration and loading compared to the lazy ETL for smaller volumes of data. However, as the total volume of data grows, the data loaded by *Óbidos* remained the lowest, compared to both eager ETL and lazy ETL. Moreover, for repeating user queries, both eager ETL and *Óbidos* outperformed the lazy ETL due to the availability

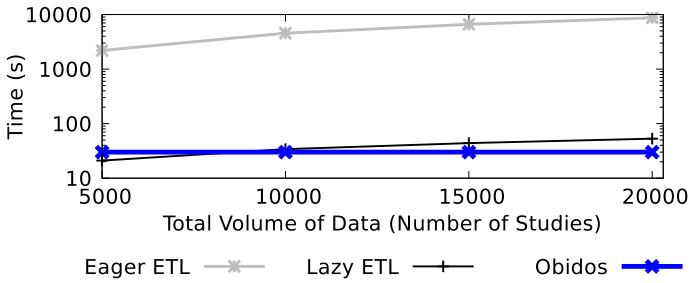


Fig. 7 Data load time: change in total volume of data sources (same user query and same replicaset)

of the integrated data repository in both eager ETL and *Óbidos*, and the storing of query answers in *Óbidos*.

The experiment was repeated for a constant total volume of data sources while increasing the number of studies of interest in the replicaset. Figure 8 shows the time taken by *Óbidos*, lazy ETL, and eager ETL to integrate and load the data from the data sources. Since the total volume remained constant, the lazy ETL and eager ETL had the same data integration and loading time, as they are oblivious to the change in the number of studies of interest. However, the performance of *Óbidos* depends heavily on how the replicasets are defined. Therefore, with the growth of the replicaset, the loading time of *Óbidos* increased. Eventually, the data integration and loading time of *Óbidos* converged with the time taken by the lazy ETL approach, as the replicaset was defined to cover all the studies in the data sources (thus, making it eagerly loading the metadata).

Finally, datasets were integrated and loaded directly from the remote data sources (such as TCIA and S3 buckets) through their web service APIs, to evaluate the effects of data downloading and bandwidth consumption associated with it. We changed the total volume of data in the data sources by adding more data to the data sources while keeping the replicaset unchanged. Figure 9 shows the time taken for *Óbidos*, lazy ETL, and eager ETL. Eager ETL performed poor as binary data had to be downloaded over the network. Lazy ETL too performed slowly for large volumes as it must eagerly load the metadata (which itself grows with scale) over the network.

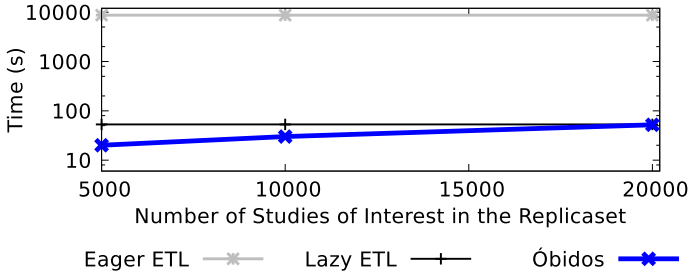


Fig. 8 Data load time: varying number of studies of interest in the replicaset (same user query and constant total data volume)

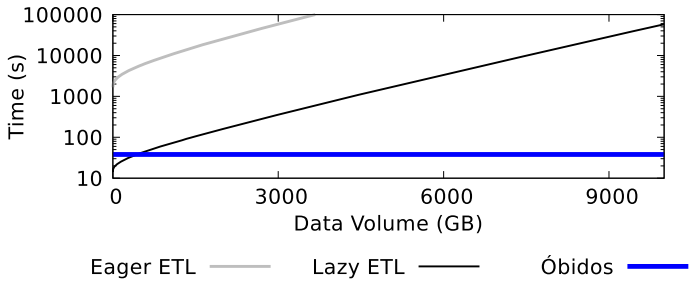


Fig. 9 Load time from the remote data sources

As with the case of Fig. 7, Fig. 9 too illustrates a fixed time for *Óbidos* data integration and loading. As the data was integrated and loaded over the Internet from the data sources, the time taken grew linearly for eager ETL and lazy ETL. However, lazy ETL consumed much lower time compared to the eager ETL. As only the datasets corresponding to the replicaset are accessed, integrated, and loaded, *Óbidos* uses bandwidth conservatively, loading no irrelevant data or metadata. Regardless of the growth of the increasing total volume of data in the data sources, *Óbidos* integrated and loaded the data at the same time as the replicaset and the user query remained the same. Therefore, the human-in-the-loop contributed positively to the integration and loading performance of *Óbidos* by narrowing down the search space from the data sources.

4.2 Performance of querying the integrated data repository

Óbidos was then benchmarked for its efficiency in querying the data and integrated data repository against the eager ETL. Query completion time depends on the number of entries in the queried data rather than the size of the entire integrated data repository. Hence, varying amounts of data, measured by the number of studies, were queried. Figure 10 depicts the query completion time of *Óbidos* and eager ETL. *Óbidos* showed a speedup compared to the eager ETL, which we attribute to the efficient indexing of the integrated data repository with the binary data with Metadata Index and the efficiency of the Data Management Layer in managing the storage and execution. The unstructured data in HDFS was efficiently queried as in a relational database through the distributed query execution of Drill with its SQL support for NoSQL data sources.

Typically, lazy ETL approaches do not consist of an integrated data repository. Therefore, we avoid comparing the query performance on the *Óbidos* integrated data repository against the lazy ETL. Eager ETL could outperform *Óbidos* for queries that access data not yet loaded in *Óbidos*, as eager ETL would have constructed an entire data warehouse beforehand. However, with the domain knowledge of the medical data researcher, the relevant datasets are loaded timely, and only those. The time required to construct a complete data warehouse would preclude any benefits of eager loading from being prominent. If data is also not loaded beforehand in eager ETL, it will consume much longer to construct the entire data warehouse before actually starting the processing of the user query. Moreover, loading everything beforehand may be

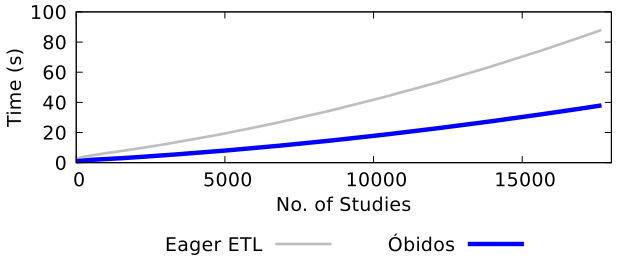


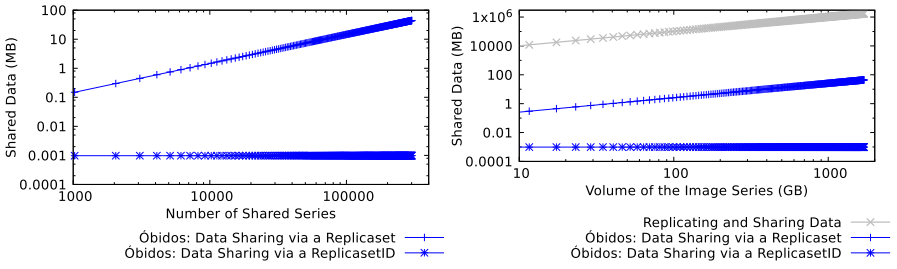
Fig. 10 Query completion time for the integrated data repository

irrelevant, impractical, or even impossible for scientific research studies due to the scale and distribution of the data sources.

Overall, in all the relevant use cases, lazy ETL and *Óbidos* significantly outperformed eager ETL as the need to build a complete data warehouse is avoided in them. As *Óbidos* loads only the relevant subsets of metadata, and does not eagerly load even the metadata, for large volumes *Óbidos* also significantly outperformed lazy ETL in its integration and loading.

4.3 Sharing efficiency of medical research data

Various image series of an average uniform size are shared between users inside an *Óbidos* instance and across multiple instances. Figure 11 benchmarks the data shared in these *Óbidos* data sharing approaches against the typical binary data transfers regarding its bandwidth efficiency. *Óbidos* can share data by sharing either the replicasetID or replicaset. ReplicasetIDs are very small and are fixed in size. Replicasets are minimal in size as pointers to actual data. However, they grow linearly when more data of the same level of granularity is shared. Negligible overhead was added in both cases as compared to sharing actual data. The *Óbidos* data sharing approach also avoids the need for manually sharing the locations of the datasets, which is an alternative bandwidth-efficient approach to sharing the integrated data. As the pointers are shared, no actual data is copied and shared. This enables data sharing with zero redundancy.



(a) With changing number of shared series **(b)** With changing volume of shared images

Fig. 11 Volume of data shared in *Óbidos* data sharing use cases versus in regular binary data sharing

The data sharing process of *Óbidos* is designed to have minimal data replication across multiple organizations, avoiding repetitive ETL efforts. Through its support for sharing datasets through a globally identifiable replicasetID, the data sharing is made efficient with minimal bandwidth overhead. Even sharing the replicaset itself was more bandwidth efficient than actually replicating and sharing the data. Furthermore, by limiting unauthorized access to the integrated data repository (through authorization mechanisms such as API keys), *Óbidos* avoids accidental sharing of confidential scientific research data. When the receiver does not have access to the integrated data repository of the sender organization, the datasets pointed by the replicaset are integrated and loaded into the receiver organization's integrated data repository.

5 Related work

Service-based data integration: OGSA-DAI (Open Grid Services Architecture-Data Access and Integration) [2] facilitates federation and management of various data sources through its web service interface. The Vienna Cloud Environment (VCE) [4] offers service-based data integration of clinical trials and consolidates data from distributed sources. VCE provides data services to query individual data sources and to provide an integrated schema atop the distributed datasets. The use of a unified schema to virtually integrate data is similar to the *Óbidos* approach. However, *Óbidos* offers a complete hybrid ETL approach and supports sharing of data with minimal data replication.

EUDAT [21] is a platform to store, share, and access multidisciplinary scientific research data. EUDAT hosts a service-based data access feature B2FIND [34], and a sharing feature B2SHARE [3]. When researchers access these cross-disciplinary research data sources, they already know which of the repositories they are interested in, or can find them by the search feature. Similar to the motivation of *Óbidos*, loading the entire data from all the sources is irrelevant in EUDAT. Hence, choosing and loading certain sets of data is supported by these service-based data access platforms. *Óbidos* can be leveraged to load related cross-disciplinary data from the eScience data sources such as EUDAT.

Lazy ETL: Lazy ETL [17] demonstrates how metadata can be efficiently used for study-specific queries without actually constructing an entire data warehouse beforehand, by using files in SEED [1] standard format for seismological research. The hierarchical structure and metadata of SEED are similar to that of DICOM medical imaging data files that are accessed by the *Óbidos* prototype. Thus, we note that while we prototype *Óbidos* for medical research, the approach is also applicable to various research and application domains.

LigDB [26] is similar to *Óbidos* as both focus on a query-based integration approach as opposed to having an entire data warehouse constructed as the first step, and it efficiently handles unstructured data with no schema. However, *Óbidos* differs as it indeed has a scalable integrated data repository, and does not periodically evict the stored data, unlike LigDB. The incremental and selective integration and loading

approach enable *Óbidos* to load complex metadata faster than the current lazy ETL approaches.

Medical research data integration: Leveraging Hadoop ecosystem for management and integration of medical data is not entirely new, and the previous work [24] indeed motivates our design choices. However, the existing approaches fail to extend the scalable architecture offered by Hadoop and the other big data platforms to create an index to the unstructured integrated data, manage the data in-memory for quicker data manipulations, and share results and datasets efficiently with peers. *Óbidos* attempts to address these shortcomings with its novel hybrid ETL approach and architecture, designed for reproducible scientific research.

Research has proposed several enhancements to data integration such as virtual data integration approaches [20] and human-in-the-loop data integration [23]. Similarly, there have been proposals such as distributed data sharing [35] aim at improving the efficiency of data sharing. However, these research approaches do not focus on big data integration for scientific research, that has its limitations as well as constraints. We narrow down our focus to scientific research, and further aim at the biomedical big data integration and sharing. Thus, we efficiently resolve the challenges in data integration and sharing, specific to the reproducible scientific research, by leveraging the human-in-the-loop.

6 Conclusion

Óbidos is an on-demand data integration system with human-in-the-loop for scientific research. It selectively integrates and loads the data and metadata in a scalable integrated data repository. By implementing and evaluating *Óbidos* for medical research data, we demonstrated the efficiency of the *Óbidos* hybrid ETL process. We presented the *Óbidos* data sharing approach to share scientific research datasets with minimal replication.

We built our case on the reality that data sources are proliferating, and cross-disciplinary researches, such as medical data research, often require access and integration of datasets spanning across the multiple data sources on the Internet. We further presented how a selective ETL approach driven by users fits well for the reproducible scientific research. *Óbidos* leverages the respective APIs offered by the data sources in accessing and loading the data while providing its RESTful APIs to access its integrated data repository. We further envisioned that various organizations with an *Óbidos* instance would be able to collaborate and coordinate to construct and share the integrated datasets internally and between one another.

As a future work, we aim to deploy *Óbidos* approach to consuming data from various scientific research data repositories such as EUDAT to find and integrate research data. Thus, we will be able to conduct a usability evaluation of *Óbidos* based on various scientific research domains and data sources. We also propose to leverage the network proximity among the data sources and the *Óbidos* instances for efficient data integration and sharing, in the future work. Thus, we aim to build virtual distributed data warehouses—data partially replicated and shared across various research institutes.

Acknowledgements This work was supported by NCI U01 [1U01CA187013-01], Resources for development and validation of Radiomic Analyses & Adaptive Therapy, Fred Prior, Ashish Sharma (UAMS, Emory), National funds through Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2013, PTDC/EEI-SCR/6945/2014, a Google Summer of Code project, and a PhD grant offered by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) under grant agreement 2012-0030.

References

1. Ahern, T., Casey, R., Barnes, D., Benson, R., Knight, T.: SEED Standard for the Exchange of Earthquake Data Reference Manual Format Version 2.4. Incorporated Research Institutions for Seismology (IRIS), Seattle (2007)
2. Antonioletti, M., Atkinson, M., Baxter, R., Borley, A., Chue Hong, N.P., Collins, B., Hardman, N., Hume, A.C., Knox, A., Jackson, M.: The design and implementation of Grid database services in OGSA-DAI. Concurr. Comput. Pract. Exp. **17**(2–4), 357–376 (2005)
3. Ardestani, S.B., Håkansson, C.J., Laure, E., Livenson, I., Stranák, P., Dima, E., Blommesteijn, D., van de Sanden, M.: B2SHARE: an open e-Science data sharing platform. In: 2015 IEEE 11th International Conference on e-Science (e-Science), pp. 448–453. IEEE (2015)
4. Borckholder, C., Heinzel, A., Kaniovskyi, Y., Benkner, S., Lukas, A., Mayer, B.: A generic, service-based data integration framework applied to linking drugs and clinical trials. Procedia Comput. Sci. **23**, 24–35 (2013)
5. caMicroscope: caMicroscope (2018). http://camicroscope.org
6. Çaparlar, C.Ö., Dönmez, A.: What is scientific research and how can it be done? Turk. J. Anaesthesiol. Reanim. **44**(4), 212 (2016)
7. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. ACM SIGMOD Rec. **26**(1), 65–74 (1997)
8. Clark, K., Vendt, B., Smith, K., Freymann, J., Kirby, J., Koppel, P., Moore, S., Phillips, S., Maffitt, D., Pringle, M.: The Cancer Imaging Archive (TCIA): maintaining and operating a public information repository. J. Digit. Imaging **26**(6), 1045–1057 (2013)
9. Dong, X.L., Srivastava, D.: Big data integration. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE), pp. 1245–1248. IEEE (2013)
10. Gradecki, J.D., Cole, J.: Mastering Apache Velocity. Wiley (2003)
11. Hausenblas, M., Nadeau, J.: Apache Drill: interactive ad-hoc analysis at scale. Big Data **1**(2), 100–104 (2013)
12. Heinzlreiter, P., Perkins, J.R., Tirado, O.T., Karlsson, T.J.M., Ranea, J.A., Mitterecker, A., Blanca, M., Trelles, O.: A cloud-based GWAS analysis pipeline for clinical researchers. In: CLOSER, pp. 387–394 (2014)
13. Hey, T., Trefethen, A.E.: Cyberinfrastructure for e-Science. Science **308**(5723), 817–821 (2005)
14. HL7: FHIR (2018). https://www.hl7.org/fhir/
15. Huang, Z.: Data integration for urban transport planning. Citeseer (2003)
16. Kadadi, A., Agrawal, R., Nyamful, C., Atiq, R.: Challenges of data integration and interoperability in big data. In: 2014 IEEE International Conference on Big Data (Big Data), pp. 38–40. IEEE (2014)
17. Kargín, Y., Ivanova, M., Zhang, Y., Manegold, S., Kersten, M.: Lazy ETL in action: ETL technology dates scientific data. Proc. VLDB Endow. **6**(12), 1286–1289 (2013)
18. Kathiravelu, P., Chen, Y., Sharma, A., Galhardas, H., Van Roy, P., Veiga, L.: On-demand service-based big data integration: optimized for research collaboration. In: VLDB Workshop on Data Management and Analytics for Medicine and Healthcare, pp. 9–28. Springer (2017)
19. Krishnan, S., Haas, D., Franklin, M.J., Wu, E.: Towards reliable interactive data cleaning: a user survey and recommendations. In: Proceedings of the Workshop on Human-in-the-Loop Data Analytics, p. 9. ACM (2016)
20. Langegger, A., Wöß, W., Blöchl, M.: A semantic web middleware for virtual data integration on the web. In: European Semantic Web Conference, pp. 493–507. Springer (2008)
21. Lecarpentier, D., Wittenburg, P., Elbers, W., Michelini, A., Kanso, R., Coveney, P., Baxter, R.: EUDAT: a new cross-disciplinary data infrastructure for science. Int. J. Digit. Curation **8**(1), 279–287 (2013)
22. Lee, G., Doyle, S., Monaco, J., Madabhushi, A., Feldman, M.D., Master, S.R., Tomaszewski, J.E.: A knowledge representation framework for integration, classification of multi-scale imaging and non-

imaging data: preliminary results in predicting prostate cancer recurrence by fusing mass spectrometry and histology. In: 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, pp. 77–80. IEEE (2009)
23. Li, G.: Human-in-the-loop data integration. Proc. VLDB Endow. **10**(12), 2006–2017 (2017)
24. Lyu, D.M., Tian, Y., Wang, Y., Tong, D.Y., Yin, W.W., Li, J.S.: Design and implementation of clinical data integration and management system based on Hadoop platform. In: 2015 7th International Conference on Information Technology in Medicine and Education (ITME), pp. 76–79. IEEE (2015)
25. Marchioni, F., Surtani, M.: Infinispan Data Grid Platform. Packt Publishing Ltd., Birmingham (2012)
26. Milchevski, E., Michel, S.: LigDB—online query processing without (almost) any storage. In: EDBT, pp. 683–688 (2015)
27. Mildenberger, P., Eichelberg, M., Martin, E.: Introduction to the DICOM standard. Eur. Radiol. **12**(4), 920–927 (2002)
28. Reichman, O.J., Jones, M.B., Schildhauer, M.P.: Challenges and opportunities of open data in ecology. Science **331**(6018), 703–705 (2011)
29. Scality: Scality RING (2018). http://storage.scality.com/rs/963-KAI-434/images/Scality%20Technical%20Whitepaper.pdf
30. Spark: Spark Framework: An Expressive Web Framework for Kotlin and Java (2018). http://sparkjava.com/
31. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proc. VLDB Endow. **2**(2), 1626–1629 (2009)
32. Vassiliadis, P.: A survey of Extract-transform-Load technology. Int. J. Data Warehous. Min. **5**(3), 1–27 (2009)
33. White, T.: Hadoop: The Definitive Guide. O'Reilly Media Inc, Sebastopol (2012)
34. Widmann, H., Thiemann, H.: EUDAT B2FIND: a cross-discipline metadata service and discovery portal. In: EGU General Assembly Conference Abstracts, vol. 18, p. 8562 (2016)
35. Zhang, Q., Zhang, X., Zhang, Q., Shi, W., Zhong, H.: Firework: big data sharing and processing in collaborative edge environment. In: 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb), pp. 20–25. IEEE (2016)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Pradeeban Kathiravelu[1,2,3] · Ashish Sharma[1] · Helena Galhardas[2] · Peter Van Roy[3] · Luís Veiga[2]

Ashish Sharma
ashish.sharma@emory.edu

Helena Galhardas
helena.galhardas@tecnico.ulisboa.pt

Peter Van Roy
peter.vanroy@uclouvain.be

Luís Veiga
luis.veiga@inesc-id.pt

1 Emory University School of Medicine, Atlanta, USA

2 INESC-ID Lisboa/Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

3 Université catholique de Louvain, Louvain-la-Neuve, Belgium