Pilot evaluation of Collection API with PID Kernel Information

Yu Luo
School of Informatics, Computing, and Engineering
Indiana University
Email: luoyu@indiana.edu

Beth Plale
School of Informatics, Computing, and Engineering
Indiana University
Email: plale@indiana.edu

Abstract

Persistent Identifier (PID) is a widely used long-term unique reference to digital objects. Meanwhile, Handle, one of the main persistent identifier schemes in use, implements a central global registry to resolve PIDs. The value of Handle varies in sizes and types without any restrictions from user side. However, widely using the Handle raises challenges on managing and correlating different PIDs for users and curators.

In this research paper, we raise an idea about the value of Handle, called PID Kernel Information, which is the critical metadata describing the minimal information for identifying the PID object. Simultaneously, an API service called Collection API, is collaborating with PID Kernel Information to manage the Backbone Provenance relationships among different PIDs. This paper is an early research exploration describing the strength and weakness of Collection API and PID Kernel Information.

1 Motivation

The motivation broadly is associating provenance with data objects in ways that are system agnostic, with enough information for basic programmatic decisions to be made, and without bogging down the resolution needed to get to an object.

The benefits of such an approach are globally associating the small information with a object, assigning a unique persistent identifier to a object and providing basic provenance information for tracking lineage of objects. This would be like associating small information with a DNS record that would reduce the number of hops to get to a machine.

We propose to associate provenance with the PID record. A tiny amount of provenance information saved in PID record would help decision progress to reconstruct the structure of lineage without getting the objects.

We assume the Handle system could be the main effort in this work since Handle system enables a flexible usage of metadata information on its schema and identifies the Data Objects rather than their locations.

We utilize the record of a handle for associating info right at the ID record. Efficiencies because of limited size but abundant provenance information. we call this info PID KI. PID Kernel Information and its potential are explored for contributing to new data ecosystem: complex data objects and their prevalence in science. How to manage the Complex Data Object into Handle System with PID KI would be a pilot research question in data diffusion.

The PID Kernel Information WG of The Research Data Alliance (RDA) organization provides the guiding principles on PID Kernel Information[1], which determine the relevant metadata elements that are suitable to embed in the PID record.

The problem in this study is attributes can’t have lists. At the high level utilization of provenance in Handle, needed for more provenance, but constrained by the guiding principles. The proper solution here is using limited provenance (backbone provenance) to maximally present the lineage.

How does one hang onto the efficiencies while staying true to the need for backbone provenance and true to the guiding principles? There is hope in a new RDA approach called collections API. Collections API is a service that defines a fine-grained collection object by the structure of Collection and Member, works as a support for reconstructing the structure of objects.

The remainder of this study is organized as follows. Section 2 presents the related work. Section 3 defines the problem space. Section 4 discusses the two data models: PID KI and Collections API. Section 5 is use cases. Section 6 describes the models of maintaining provenance in Handle and Collection API. The experiment architecture and environment of the experiment is presented in Section 7. Section 8 presents the result of Handle and Collection API study. Section 10 Open Questions completes the work.
2 Related Work

The academic and industry already develop similar Persistent Identifiers with different semantic and syntax. Paskin [2] pointed out the high level understanding about the DO of the scope, the syntax and the resolution. In his research, he raised up a definition for metadata, DOI Kernel, the minimum set of metadata information specified by the IDF registered with DOI. The DOI Kernel is similar to the PID Kernel Information that we described in this research, excepting that PID Kernel Information contains the Backbone Provenance to enhance the connections to other objects for drawing the Provenance Network. DOI tends to provides abundant information for users to understand the objects.

Archival Resource Key (ARK) [3] presented by Kunze is another approach to make the permanence of object. The ARK is not like other conventional schema (URL, PURL, and URN), it provides three things: object, metadata and provider’s commitment. While adding '?' to the end of one ARK link, the server would return the human-readable information, for example, the Electronic Resource Citation using Dublin Core Kernel metadata, including who, when, where and what. All these kernel information would help users just understand the object, but not the relationships to other objects.

The PID Kernel Information of Handle in this research inherits the benefit of Persistent Identifier and try to add more efforts to the data ecosystem, not only referencing to the single object, but also presenting the entire graph of connected objects.

Some other researchers raise the idea about network of provenance, the visualization of connected objects. Zhou [4] discussed their work, ExSPAN, a framework that achieves the network provenance at Internet-Scale. The ExSPAN stores the provenance in a relational table, and try to make a granular relationship graph of objects for efficient query and maintain. The idea of keeping network provenance is quite similar to the research of this paper, but ExSPAN is working on the local use and analysis. However, the research in this paper is forcing on the global scope that building the provenance network for objects, offering efficient approach on publishing and maintaining public objects.

Chen [5] provides a approach to visualize the graph of network provenance captured from streaming data. The research is fully maximizing the benefit of provenance, and use visualization techniques to interact with data object. The idea of representing the network data by using provenance from Chen’s research is a good reference to start our research, however, our research wants to make the the Internet Scale of presenting and sharing data, using the minimal metadata information, keeping fine-grain provenance relationship, by using Persistent Identifiers. All these requirements are challenges for current Network Provenance or Provenance Network representation work, and this is the motivation that we do the pilot evaluation on Handle System and Collection API Service, attempting to contribute to the Publishing Ecosystem.

A visualization system, Komadu [6], is a tool embedded in other softwares to ingest and present the provenance information while doing workflow. The system is powerful on providing high level understanding of provenance among different object. However, the centralized database and local sharing mechanism restricted the data sharing on Internet.

3 Problem definition

The goal of this study is finding a proper way of using Handle and Collection API to maintain the lineage of objects with provenance information, and sharing these provenance information and object relationships on Internet. The questions here are how to balance the provenance in both systems and how to distribute the resolving workloads between them.

We assume that data are in the form of Data Objects. Each Data Object is identified by a PID. A data object can be complex on a number of dimensions [7]: multi-format, multi-structure, multi-source, multi-model and multi-version. For our study, a Data Object is discrete - that is, a data object cannot be decomposed, but can be related to other data objects.

3.1 Backbone Provenance

Another important concept is Backbone Provenance. Backbone Provenance as a exploratory idea is raised in PID KI. The Provenance Working Group of World Wide Web Consortium (W3C) has defined and published a fine-grain world-wide language for exchanging provenance information [8] among different applications. W3C PROV gives us a clear understanding and guide on using provenance relationship among different objects.

Backbone Provenance is the essence of the provenance relationship among provenance objects. We endorse Backbone Provenance exists in PID KI. All Backbone Provenance is embedded in the PID KI as key-value pair as other normal PID records including [wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternateOf and specializationOf]. These six provenance elements could fully described the provenance relationship among Entities. Full provenance can be a superset of Backbone Provenance. That is, full provenance can replicate the Backbone Provenance, but it doesn’t replace Backbone Provenance in PID KI. Also, as a pilot evaluation, we just select these fields for study purpose.

Provenance can be visualized as a set of data objects as the nodes of a graph where the nodes are linked by provenance relationships.

The comparison between the graph definition of W3C PROV [8] and Provenance relationship is shown in Fig. [9]
provenance relationship only contains the entities and the entities are connecting with Backbone Provenance and Had-Member provenance.

Figure 1: High level overview of Provenance relationship

Backbone provenance is a limited set of W3C PROV definitions, but it is sufficient to represent the relationships between entities, especially the file objects in collection model.

3.2 Handle system

Handle System underpins the technical infrastructure of DOI, enables a flexible usage of metadata information on its schema, identifying the Data Objects rather than their locations. Key-to-value paired entry is the data architecture involved in Handle (PID) records. Unfortunately, identifying an Data Object is more difficult than referencing to its current location.

While showing the identification of a Data Object, Handle takes responsibility for the integrity of Data Object. Neither Handle nor DOI offers a flexible strategy for managing Data Objects. Robustly using the infrastructural schema of Handle would be a choice, but using Models would gain a fine-grain structure of Data Object.

3.3 PID Kernel Information

The key-value pair of Handle records in LHS is added into PID Kernel Information. PID Kernel Information would not fully describe the object, the purpose of PID Kernel Information is making smart programmatic decisions though inspection of PID records alone.

Handle system [9] is a general-purpose global name service for assigning Persistent Identifiers (Handles) to Data Object, and resolving the identification of the resource that Handle referenced.

The two components of the Handle System [10] are the Global Handle Registry (GHR) and the Local Handle Service (LHS). Global Handle Registry manages the authority Handles and provides the service information of any Local Handle Service. The naming authority requests from client would be directly sent to the GHR, and GHR sends back the service information. Then client would follow the service information sending the request to responsible LHS.

Local Handle Service controls a restricted namespace that manages Handles under given naming authority. Accepting the requests from client, LHS sends the Handle results (PID Kernel Information) back to client.

The Handle (PID) is made up with two parts: prefix and suffix, for example, 20.500.12033(prefix)/test.demo(suffix). The value of Handle uses a common data structure, including index, type, data, permission, TTL (time-to-live) and timestamp. A pre-defined Type in DTR (Data Type Registry [11]) that describes the semantics and syntax of data in data field is a cogitative improvement for global using and defining an acceptable protocol for shared data.

PID Kernel Information is a set of attributes of describing a object saving in the Handle record. In this study, we only keep the six provenance fields out of the entire PID KI, which are [wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternateOf and specializationOf]. The values for these fields are the links to the Handle or Collection API objects.

3.4 Collections API

The Collection API Service [12] (C-API-S) is a prototype product of the Research Data Alliance Research Data Collections Working Group [13].

The service defines a fine-grained collection object by the structure of Collection and Member. Collection capabilities, collection properties and collection membership compose to the first level of data hierarchy, describing the full actions and essential metadata of the collection. Membership of a collection builds the graph with the relative Member objects, linking to the next level of collection entry. The second but last layer of hierarchy is the Member entry, which is divided into general properties of location, description, data type and ontology. Note that a collection could be the member of another collection, inheriting the logic of tree structure.

Collection-Member architecture could be extended to a hierarchy tree structure, which has infinite possibility on the structure of Digital Data Object. The flexibility of the Collection API service is trying to overcome the challenge of inconsistent structure of Data Object before and after processing, which assist Handles (PID KI) in defining the complex structure of Data Object.

4 Mapping between Collections API and Backbone Provenance

The mapping between the Collections API data model and the model of Backbone provenance and its embedding in the Handle record forms one of the contributions of this paper.

A Collection is quite different from a Directory or Folder. In file systems, the folder/directory defines the collection of files or other folder/directory. The collection, however, is
an object that references other objects (members/collections) through a provenance relationship. So the normal directory/folder could be subset of the Collection; the object referencing to other objects by 'HadMember' or Backbone Provenance. Therefore, the connected objects could be extended to a directed acyclic relationship with the relationships connecting them, like the Computer relationship that each node could directly connect to other nodes. In this case, collections and their referencing objects are the nodes, the provenance relationships are the connections between them, then all of them make up the Provenance relationship, the directed acyclic Provenance relationship.

Overall, one collection and its relationships could be interpreted as the subset of provenance relationship, and one collection could extend to the entire Provenance relationship if we trace its relationships till end. However, in this research we define the simple case that each collection referencing to member objects with 'HadMember' relationship, and collections and members could have their Backbone Provenance to other objects. The relationship could be seen in Fig. 2.

The Provenance relationship is directed and acyclic result from the features of provenance: timeliness and directionality. Timeliness indicates the order of data object joining the Provenance relationship. In the Provenance relationship, we first publish one object as root, and then published other objects later. The publishing date is a hidden information showing that we cannot find a unpublished data before a published data in Provenance relationship. Directionality indicates the relationship among objects based on timeliness. For example, we could generate a next version object based on the current object, but inverse doesn’t work. Directionality is the visualization of timeliness with actions of the object.

5 Use Cases

In this section, we present three possible approaches in three use cases to represent data objects in the PID Kernel Information with the assistant of Collection API Service. "HadMember" link is supported by Collection API service, helping to achieve the integration of the complex data object in data model of PID KI and C-API-S object. In these three solutions, C-API-S would basically handle “HadMember” provenance relationship for all complex data objects. Then we gradually transfer the Backbone Provenance information into C-API-S object in different structures, testing the collaboration between PID KI and Collection API service on managing and resolving Provenance relationship.

We define several synthetic use cases used for experimental compassion. A provenance use case is represented as a directed acyclic Provenance relationship shown in Fig. 2 of vertices (node) and edges (relationship), N = \{V, E\} where a vertex (node), V corresponds to data object, edge (relationship), E is directed acyclic provenance relationship between vertices.

Synthetic use cases are developed for experiment purpose. We manually control the structure and relationship of Complex data object, building use cases for testing different data structures in same environment.

1. Directed linear backbone relationship, spreading with HadMember on branch:

Linear Backbone Provenance relationship contains 100 collection data objects. Each collection object points to 4 member objects with 'HadMember' relationship. And collection objects could be connected by the Backbone Provenance including [wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternateOf and specializationOf]. The data structure is shown in Fig. 3 with T(x) indicates timeliness and arrow presents direction.

2. Directed acyclic backbone relationship, spreading with HadMember on branch

Directed acyclic Provenance relationship contains 100 collection data objects. Each collection object connects 4
member objects by 'HadMember' relationship. And collection objects could be connected by [wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternativeOf and specializationOf]. In this case, we put more Backbone Provenance in the relationship among collections than case 1. One collection object could have more than one directed acyclic Backbone Provenance to other collection objects. The data structure is shown in Fig. 4 with timeliness and directionality features.

Figure 4: Directed acyclic backbone relationship, spreading with HadMember on branch, collections connects to collections

3. Directed acyclic backbone relationship, spreading with HadMember on branch (Collections and Members connect to same type objects):

This synthetic use case is marked as case 3, developed based on the use case 2 by adding member-to-member provenance relationship including [wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternativeOf and specializationOf]. This case contains 100 collection data objects. Each collection object connects 4 member objects by 'HadMember' relationship. And collection and Member objects could be connected by [wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternativeOf and specializationOf] to same type objects. In this case, one collection or member object could have more than one directed acyclic Backbone Provenance to other collection or member objects. The data structure is shown in Fig. 5 with timeliness and directionality features.

4. Directed acyclic backbone relationship, connected by Backbone Provenance including HadMember relation:

Directed acyclic Provenance relationship contains 500 data objects (100 collection objects and 400 member objects) with directed acyclic Backbone Provenance among them. Still, one collection object and its member objects are connected by 'HadMember' relationship, and collection object and member object could connect to different type object with Backbone Provenance in this provenance relationship graph. The data structure is presented in Fig. 6 showing the timeliness and directionality of provenance of objects.

Figure 5: Directed acyclic backbone relationship, spreading with HadMember on branch, Collections and Members connect to same type objects

Figure 6: Directed acyclic backbone relationship, connected by Backbone Provenance including HadMember relation

6 Proposed Solutions

Handle System and C-API-S both take responsibility of saving provenance information in their objects. The differences between proposed solutions are the strategies of balancing provenance in PID Kernel Information and C-API-S objects.

S1: most of Backbone Provenance saved in PID KI except HadMember provenance in C-API-S

Handle System is in charge of saving and presenting the data objects with the connections. The PIDs are the vertices/nodes in the provenance relationship. Backbone Provenance relationships are maintained in Kernel Information as edges in relationship, keeping 'HadMember' provenance within C-API-S model. Only accessing Handle System and resolving PID KI, consumers could construct a large proportion of provenance relationship. However, if two objects are connected by 'HadMember' provenance, the resolving client would query C-API-S to construct the rest of provenance relationship which is maintained in Collection-Member data model, pointing to the PIDs of that C-API-S objects.

S2: PIDs and C-API-S Member objects are the vertices in Provenance relationship, both of them contains the Backbone Provenance

Instead of saving the entire Backbone Provenance rela-
tionship within PID Kernel Information, solution 2 enforces
the C-API-S Member objects to ease the burden of PIDs,
separating the provenance relationship from PIDs into C-
API-S member objects, transferring the workload of resolv-
ing Provenance relationship from Handle System to C-API-
S. While resolving the objects, client would first access the
PIDs to construct the PIDs’ provenance relationship, and
then access the C-API-S to acquire the provenance within
C-API-S Member objects to next Handle objects. Finally,
combing the Provenance relationship of PIDs and C-API-S
Members object into the integrated relationship.

S3: C-API-S maintains the Backbone Provenance

The strategy of S3 is simple, directly using the Collection-
Member model of C-API-S to maintain the provenance in-
formation, but exploratory, replacing PID’s role with C-API-
S objects in Provenance relationship to evaluate the strength
and weakness of this exploration. In resolving process, client
would only access only one PID to start the workload, and
then it would iteratively access the C-API-S and retrieve the
provenance information to construct the provenance relation-
ship.

7 Experimental Setup and Architecture

We proposed to evaluate the strength and weakness of Han-
dle System and C-API-S considering the proposed solutions
and use cases in below section with proper workload and
evaluation metrics.

We launch one test instance on Amazon AWS instance
with 4 Core CPUs and 16 GB RAM to host the Handle Sys-
tem and C-API-S.

There is no conflict in the communications between Han-
dle System and C-API-S. For both servers, they would pro-
cess the requests in order, and there is no communications
occurred between Handle System and C-API-S. In normal
workflow, the resolving client would first acquire PID KI
with provenance information from the Handle System, then
following the provenance track, the client would acquire the
Collection or Member object with provenance information
from C-API-S. In some cases, the query requests would re-
peatedly send to same service over a period of time for ob-
taining the same type objects based on the use case we used
in experiment.

Architecture details of C-API-S are presented in Fig. 7.
C-API-S is implemented in Python with Flask HTTP Frame-
work. The request is passed from HTTP interface to RDF
database though layered components. DB interface maps the
collection-member model into graph-based RDF entries in
the communication with SPARQL endpoint.

7.1 Workload Description

In the experiment, we proposed three tests for three proposed
solutions to evaluate the performance of Handle System and

Test 1: the graph would be built by acquiring PIDs with
‘HadMember’ related objects

The workload simulates the environment of C-API-S and
Handle System. With the pilot attempt of C-API-S, we em-
bed it with light workload in case of affecting the perform-
ance of Handle System lightly.

We assign PIDs for all data objects, and each PID KI con-
tains a block of provenance information with wasDervied-
From, wasRevisionOf, wasQuotedFrom, hasPrimarySource,
alternateOf and specializationOf. The Backbone Provenance
is drawn by PIDs with hadMember provenance within Mem-
ner Objects linking their PIDs.

The workload simulates the HTTP GET queries from re-
solving client to retrieve Backbone Provenance in PID KI
from Handle System with a hadMember link referencing to
Collection API Member objects. The client issues another
HTTP GET query to retrieve 4 member objects, and then
access the PID KI of them from Handle System to acquire
Member objects’ provenance. At this point, we obtain the
Provenance relationship of one collection object. The client
repeatedly issues the HTTP GET to traverse through the
provenance relationship to reach the endpoints. The work-
load diagram is demonstrated in Fig. 8.

Test 2: the graph would be built by acquiring PIDs and
Member objects with fine-grained relationships

The Test 2 balances the workload between Handle System
and C-API-S. Comparing to Test 1, the C-API-S takes more
workloads in building the provenance relationship. Moving
partial Backbone Provenance from PID KI to Member Ob-
ject, drawing Backbone Provenance with both PIDs and Col-
le collection API member object, and finally skipping the step of retrieving PID KI of retrieved Member objects.

The workload consists of HTTP GET queries from resolving client to retrieve PIDs and Collection API Member objects. Once the client acquires the content of Collection API member objects, it will find a block of provenance information with wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternateOf and specializationOf, which is same as the provenance block in PID KI. All these provenance links within Member object provenance block are the references to the PID KI of the next node. By the time, HTTP GET queries have accessed the PID KI and Member object once to get the provenance relationship of one entire collection object. Further, the client traverses though the edges of relationship to reach the endpoints and draw the provenance relationship. The workload diagram is demonstrated in Fig. 9.

![Figure 9: PID and member object graph retrieving workload diagram](image)

Figure 9: PID and member object graph retrieving workload diagram

Test 3: the graph would be built by acquiring Collection and Member objects of C-API-S

In the last exploration, the C-API-S maintains all provenances instead of PID KI. Moving all Backbone Provenance from PID KI to Member Object, drawing Backbone Provenance relationship only within Collection API Member object and Collection API Collection objects, and skipping the step of retrieving PID KI.

HTTP GET queries are simulated by resolving client to retrieve data objects in C-API-S. Both Collection object and Member object contains a block of provenance information with wasDerivedFrom, wasRevisionOf, wasQuotedFrom, hasPrimarySource, alternateOf and specializationOf. The communications are issued within C-API-S, building the provenances relationship by collection nodes and member nodes. The workload diagram is demonstrated in Fig. 10.

![Figure 10: collection and member object graph retrieving workload diagram](image)

Figure 10: collection and member object graph retrieving workload diagram

8 Experimental results and Analysis

There are two kinds of experiments in this section: Baseline Experiment and Non-baseline Experiment. Before doing these two experiments, we also make a experiment for depositing the data into environment and collect the runtime metrics of throughput and cost.

8.1 Depositing Data

The preparation of the experiment is depositing data objects into the Handle System and C-API-S for testing the registering performance: PID Object, Collection Object and Member Object. As we described above, we have three use cases, So for each solution, we would totally deposit 15000 objects, and more details are presented in table 1.

| Case | PID Object | Collection Object | Member Object |
|------|------------|-------------------|--------------|
| 1    | 9000 objects | 1500 objects      | 6000 objects |
| 2    | 9000 objects | 1500 objects      | 6000 objects |
| 3    | 9000 objects | 1500 objects      | 6000 objects |
| 4    | 9000 objects | 1500 objects      | 6000 objects |

Table 1 Numbers of Data Objects deposited into C-API-S and Handle System
All these data objects are registered from client side by using three solution approaches in different periods. 66000 registration requests (36000 PID objects, 6000 Collection objects, 24000 Member Objects) are sequentially sent by client. Meanwhile, client records the cost of entire registration of each request. The registration costs are listed in the Fig. 11. Comparing the registration cost of Handle System and C-API-S from client side, we observe that the Handle System consumes around 31 milliseconds and C-API-S consumes around 54 milliseconds to register the objects.

For addition, we observe the CPU and Memory usage for both services while processing the depositing requests, shown in Fig. 12. From the observation, the CPU usage of Handle System and C-API-S has three periods which matching the process of registering objects for three solutions. Also, C-API-S costs more CPU but less Memory resources than Handle System since C-API-S requires computing resources on validating the schema and data properties.

In the Server side, we also observe the internal cost metrics of registering an object in Handle System and C-API-S, shown in Fig. 13. The C-API-S costs 49.6 milliseconds on Collection Objects registration and 49 milliseconds on Member objects registration. The Handle System costs 8 milliseconds on registering the objects.

8.2 Resolving Algorithm for Handle Object and C-API-S Object

In resolving client, the HTTP Get request is sent to REST API Servlet of Handle System, calling Berkeley Database to get the handle value, and to the HTTP Interface of C-API-S, calling RDF database (provided by Apache Marmotta) to get the Collection/Member values. After resolving, client acquires the values from Handle System and C-API-S, it will extract the provenance block, resolving the PID or C-API-S URL to relative services recursively. The recursive algorithm of traversing the backward Provence relationship started from the original PID is presented in Algorithm. 1.

8.3 Baseline Experiment

In Baseline Experiment section and below section, we are doing the experiments on resolving the data objects from C-API-S and Handle System based on the different solution approaches. The purpose of representing the data is evaluating the strength and weakness of different data structures in
Algorithm 1: Resolving Algorithm for Handle objects and C-API-S objects

input: Original PID of the Handle Object
output: The Provenance relationship of Original PID

1: URL ← OriginalPID;
2: function GETPROVRELATIONSHIP(URL)
3: ID ← Empty;
4: Provenance ← Empty;
5: if isPID(URL) then
6: content ← callHandle(URL);
7: ID ← getID(content);
8: Provenance ← getProv(content);
9: else
10: content ← callCOLLAPI(URL);
11: ID ← getID(content);
12: Provenance ← getProv(content);
13: for prov: value ∈ Provenance do
14: if !isEmpty(value) then
15: setProrelationship(ID, prov, value);
16: GetProvrelationship(value);

8.3.3 Querying Workload in Server

We simulate the synthetic use cases as real world data, and the querying workloads of case 2, case 3 and case 4 cannot access all objects in Provenance relationship since we randomly define the provenance relationship among objects. Totally, we send 85730 requests to the Handle System and C-API-S, excluding the relationship latency, observing the internal cost of processing queries in both servers, more details are presented in Table 2 and Fig. 14

Figure 14: Resolving Cost of Handle objects and C-API-S objects in Server

In the Table 2 we list four request types: Members, Member, Collection and Handle. Members is a C-API-S query request to get 4 member objects of a collection. Member is a C-API-S query request to get 1 member object of a collection. Collection is a C-API-S query request to get 1 collection. And Handle is a Handle System query request to get 1 handle value.

The analysis of the experiment could be started on same case in different solution approaches. In this runtime metric, we got 12 results based on 3 cases and 3 solution approaches. For each case, we apply 3 solution approaches on managing the data structure in Handle System and C-API-S.

Overall, the C-API-S cost on resolving in much higher than the Handle System. The average Handle resolving cost is less than 1 ms. And resolving cost of any objects in C-API-S is more than 1 ms.

Case 1: Directed linear backbone relationship, spreading with HadMember on branch:

In case 1, the different performances of C-API-S and Handle System will determine the best approach. The first observation is the total time cost of different solutions to query case 1 dataset. The solution 2 cost the least time on all three solutions to represent the complex data object. The advantage of solution 2 is less Handle query in its entire workloads. Solution 1 processes 2500 Handle objects, 500 for collection objects and 500 × 4 member objects, which is more than solution 2. Even though solution 3 processes 500 Collec-
Table 2 Summary of Resolving workloads in Server

| Case | Solution 1 | Solution 2 | Solution 3 |
|------|------------|------------|------------|
|      | Members    | Handle     | Members    | Member     | Handle     | Collection | Members | Member |
| size | 500 objects| 2500 objects| 500 objects| N/A        | 500 objects| 500 objects| 500 objects| N/A     |
| cost | 42.12 ms   | 0.38 ms    | 35.84 ms   | N/A        | 0.62 ms    | 10.32 ms   | 33.96 ms | N/A     |
| total| 22010 ms   | 18230 ms   | 18348.69 ms| 15423 objects| 15136 objects| 15136 objects| 15136 objects| 15136 objects|
| size | 463 objects| 2315 objects| 463 objects| N/A        | 463 objects| 463 objects| N/A     |
| cost | 34.39 ms   | 0.2 ms     | 32.01 ms   | N/A        | 0.33 ms    | 8.55 ms    | 31.08 ms | N/A     |
| total| 16385.57 ms| 14973.42 ms| 18348.69 ms| 15423 objects| 15136 objects| 15136 objects| 15136 objects| 15136 objects|
| size | 287 objects| 16571 objects| 287 objects| 15136 objects| 287 objects| 287 objects| 15136 objects| 15136 objects|
| cost | 33.46 ms   | 0.18 ms    | 31.2 ms    | 6.98 ms    | 0.19 ms    | 8.28 ms    | 30.79 ms | 6.51 ms |
| total| 12583.8 ms | 117334.1 ms| 159748.4 ms| 15136 objects| 2129 objects| 2129 objects| 2129 objects| 2129 objects|
| size | 417 objects| 4214 objects| 417 objects| 2129 objects| 417 objects| 417 objects| 2129 objects| 2129 objects|
| cost | 33.78 ms   | 0.17 ms    | 31.48 ms   | 6.63 ms    | 0.18 ms    | 8.35 ms    | 30.92 ms | 6.57 ms |
| total| 14802.64 ms| 27700.71 ms| 30363.12 ms| 2129 objects| 2129 objects| 2129 objects| 2129 objects| 2129 objects|


tion objects of C-API-S, Collection object of C-API-S needs more time than Handle object on querying.

Solution 2 benefits from all aspects of workload throughput and time cost on Complex Data Object representation. A early conclusion for this subsection is that Handle System is the best choice for presenting single and simple data object with simple provenance relationship. And saving provenance information into Member object of C-API-S with All Member querying functionality could be a proper choice to deal with little complex data structure.

**Case 2: Directed acyclic backbone relationship, spreading with HadMember on branch (only Collections connect to Collections):**

Working on the case 2 for three solution approaches, we get the same conclusion that Handle System is the best choice to present the single object. Service will take huge advantage on resolving the object within Handle System result from its good architecture and database design.

Further, both case 1 and case 2 show the solution 2 is the best choice to handle the datasets because of less Handle requests. Considering the solution 2 architecture, it puts the provenance relationships in Member object within C-API-S instead of using Handle for that Member object to store provenances. When we resolve in one C-API-S member object, we could get all provenance information of it, there is no need to go through the Handle object. For example, if one Member object has no provenance relationship, we can know it and don’t query Handle object; Otherwise, we could prepare to query the Handle System.

Limited to the dataset we use, we only set the provenance relationship in collection-to-collection level objects, so the result is similar to the experiment of case 1. In the next subsection, we are testing the case adding member-to-member Backbone Provenance relationship to observe more information.

**Case 3: Directed acyclic backbone relationship, spreading with HadMember on branch (Collections and Members connect to same type objects):**

The case 3 shows different result since we make more complex data structure based on case 2. In the result, we first observe the 1 Member object querying result on Solution 3 column. The average cost of querying 1 Member object is less than querying 1 Collection object, and 4 times of the cost of 1 Member object querying is still less than cost of querying all Member objects. We assume the cost of 1 Member object querying should be equal to the average cost of 4 Member objects querying. Actually, it is not because of different query algorithm. All Member objects querying would take less time on database querying by using the key of the Collection object rather than 4 × of 1 Member querying, especially on data existing check and number of database accessing. In this experiment, we set 1 collection has 4 member objects. Obviously, 4 Member objects querying costs more time since it gets 4 objects from database than other C-API-S querying operations, but has better performance on its average for each object. We conclude that one query of all Member object is not appropriate but only approach to handle a fine-gain provenance relationship between Collection and Member objects.

The case 3 on Solution 3 help us acquire a clear understanding about the C-API-S, its weakness and strength: 1) performance order of functionalists to represent objects is: 1 Member > 1 Collection > All Members. 2) All of three query functionalists in C-API-S are less efficient than Handle querying functionally. 3) C-API-S has three types of data object and could manage clear fine-grain relationships in them.

Overall, case 3 gives us a different result that solution 1 maximizes the advantage of Handle System comparing to Solution 2 and Solution 3. Solution 2 and Solution 3 spend too much time on presenting the data structure based on C-API-S architecture design.

One more thing about the C-API-S All Member objects querying. We set a collection has 4 member objects, and average cost of All Members querying is better than 1 Member querying. If the collection has more member object, the average cost of All Members querying will be much lesser than 1 Member object querying. Inversely, if a collection has fewer number of member object, the closer of the average cost of

10
All Members querying to 1 Member querying.

Directed acyclic backbone relationship, connected by Backbone Provenance including HadMember relation

The case 3 use case is the best one that simulates the real world case, applying the very complex provenance relationships in data structure. This use case result finally provides the advantage of Handle System on representing the data objects. Three solution approaches all take same dataset, varying on the resolving algorithms of accessing different amount of Handle objects and C-API-S objects. Solution 1 is the best that it costs less on accessing the largest number of Handle objects. The Solution 2 is better than Solution 3 that both of them access same amount of 4 Member objects and 1 Member object, but accessing Handle System assists solution 2 costs less time on representing complex data. However, the result would be varied based on the sample we use, for example the case 3 has more provenance relationships in objects than case 4. Therefore, the percentage of Handle object and Collection object in representing graph would result in the different costs, but still present the strength and weakness of Handle System and C-API-S obviously or less obviously.

8.4 Non-baseline Experiment

Addition to the Baseline Experiment, we observe the runtime metrics from client, caring about how to represent the Provenance relationship for consumers. The Fig. 15 shows the workloads in each case and its result. The Table. 3 summarizes the resolving work for all workloads.

Figure 15: Resolving Cost of Handle objects and C-API-S objects in Client

The resolving result from client is to observer the relationship communicating cost between server and users except the relationship latency. The results of 12 small experiments of Non-baseline Experiment, 3 solutions × 4 use cases, is not different from the Base-line Experiment we observed, what we care here is the communication cost of transferring different objects. Handle object and C-API-S object are different types with different sizes, especially the Collection object and Member object in C-API-S, so the representing processes would cost different time on receiving the output from server to client. We conclude the communication cost of this experiment on Table 4.

We suppose the size of different type data objects would not have significant effects on the communication cost, since in the experiment, we only put few bytes data (provenance links) into the Handle object and C-API-S object. The Handle System takes 1.985 milliseconds to transfer the data to client. C-API-S takes 3.75 milliseconds, 2 × Handle object communication cost, to send the Collection object to client. However, the communication costs between All Members (4 Member objects) and 1 Member are similar, transferring All Members (4 Member objects) to users costs 4.04 milliseconds and transferring 1 Member to users costs 3.645 milliseconds. Hence, we can finally conclude that the size of C-API-S object would not affect the communication performance of C-API-S. But, one more conclusion is raised that All Member objects would take advantage on transferring the data, that in client side we prefer use All Member querying to get all data objects.

8.4.1 Extra Functionality in C-API-S: GetProperty

In C-API-S, the server provides other flexible functionalists, like joining two collection objects, finding intersection of two collection objects, finding matched collection objects, and returning property of member objects. Here, we do the special experiment on use case 3 to test the getting property of Member objects function. The getting property functions uses the same algorithm of querying Member object, reducing the content on transferring. The Fig. 16 displays the Query Cost from C-API-S server and client. The server side average cost is 6.44 milliseconds, the client side average cost is 9.95 milliseconds, and the average communication cost is 3.51 milliseconds.

Figure 16: Resolving Cost of Getting Property from C-API-S

8.4.2 Time Cost on C-API-S Querying

The time cost of querying C-API-S insistsents of two main parts: database processing and DB Interface processing. The
Table 3 Summary of Resolving workloads in Client

| Solution | Handle | Collection | Members | Member | Members | Member |
|----------|--------|------------|---------|--------|---------|--------|
| **case 1** |  |  | | | | |
| Members | 500 objects | 500 objects | N/A | 500 objects | 500 objects | N/A |
| size | 1.94 ms | 39.94 ms | N/A | 2.33 ms | 14.1 ms | 37.89 ms | N/A |
| average cost | 27955 ms | 21135 ms | 25995 ms | N/A | 21830.45 ms | N/A |
| total | | | | | | |
| **case 2** |  |  | | | | |
| Members | 463 objects | 463 objects | N/A | 463 objects | 463 objects | N/A |
| size | 1.63 ms | 35.97 ms | N/A | 1.79 ms | 12.25 ms | 34.9 ms | N/A |
| average cost | 21529.5 ms | 17399.54 ms | 21830.45 ms | N/A | 10.25 |
| total | | | | | | |
| **case 3** |  |  | | | | |
| Members | 287 objects | 16571 objects | N/A | 15136 objects | 15423 objects | N/A |
| size | 1.65 ms | 35.77 ms | N/A | 1.68 ms | 12.22 ms | 35.35 ms | N/A |
| average cost | 38213.71 ms | 273940.2 ms | 15136 objects | N/A | 10.37 |
| total | | | | | | |
| **case 4** |  |  | | | | |
| Members | 417 objects | 4214 objects | N/A | 2129 objects | 2546 objects | N/A |
| size | 2.13 ms | 35.35 ms | N/A | 1.592 ms | 11.93 ms | 35.3 | |
| average cost | 40297.52 ms | 40792 ms | | | | |
| total | | | | | | |

Table 4 Summary of relationship Communication Cost from Server to Client

| Solution | Handle | Collection | Members | Member | Members | Member |
|----------|--------|------------|---------|--------|---------|--------|
| **case 1** |  |  | | | | |
| Average Communication Cost | 1.56 ms | 4.09 ms | 1.71 ms | 4.1 ms | N/A | 3.78 ms | 3.93 ms | N/A |
| **case 2** |  |  | | | | |
| Average Communication Cost | 1.43 ms | 3.96 ms | 1.46 ms | 3.96 ms | N/A | 3.7 ms | 3.82 ms | N/A |
| **case 3** |  |  | | | | |
| Average Communication Cost | 1.47 ms | 4.42 ms | 1.51 ms | 4.17 ms | 3.86 ms | 3.94 ms | 4.56 ms | 3.74 ms |
| **case 4** |  |  | | | | |
| Average Communication Cost | 1.36 ms | 4.01 ms | 1.41 ms | 3.82 ms | 3.48 ms | 3.38 ms | 3.56 ms | 3.5 ms |

Database processing is ingesting data from RDF database, and DB Interface processing is converting data type from graph to C-API-S object. The Fig. [17] highly indicates the time cost in these two layers, and here is the reason why C-API-S cost more time than Handle System.

![Figure 17: Time Cost when Querying C-API-S](image)

9 Discussions and Summary

This paper is not working on a research about what the Complex Data Object is or how to manage the Complex Data Object, it is doing a pilot evaluation of the strength and weakness of Handle System and C-API-S, and how they can collaborate to manage and represent the Provenance relationship of Complex Data Object.

In this pilot evaluation of C-API-S and Handle System on handling Complex data object, we simulate 4 use cases with 3 proposed solution approaches to test the different data structures of managing data object in Handle System and C-API-S.

In Handle object, we are trying to keep minimal kernel information within its records, without any complex contents, like lists of member information, to make easy reading, parsing, transferring and understanding. With this definition, Handle System would be the main component to support the Provenance relationship. The strength of Kernel Information is also the weakness of the Handle System that it cannot handle the complex provenance relationship, like ‘HadMember’ relationship. That’s why we are doing the pilot evaluation on C-API-S.

C-API-S presents a fine-grain relationship of objects by using Collection-Member data model. Collection-Member model provides an easy way to handle the ‘HadMember’ relationship without violating the definition of PID Kernel Information. However, the Collection-Member data model of C-API-S has some disadvantages. C-API-S gains the benefit from Collection-Member model, presenting a clear graph of objects, meanwhile paying the cost of more time on data managing and querying. We rely on C-API-S to help Handle System deal with complex relationships at minimal cost.

These 12 experiments tell us more information about the collaboration work between Handle System and C-API-S, showing the undiscovered phenomenons, helping us make conclusions to manage and represent the Provenance relationship of Complex Data Objects.

Based on the use case we designed, we could classify...
the Provenance relationship into three categories: collection-to-collection, type-to-type and all-to-all. The collection-to-collection means the collection object has relationship to other collection object, like copying folder from other location, ignoring the member objects of them since member objects would not connect to other objects. The type-to-type means only same type object would connect to each other, collection object to collection object and member object to member object, like updating the content of a folder and also make clear relationship to describe the provenance of member objects. The all-to-all means all objects could connect to others, no matter collection object or member object, like combining files from different folders. These three categories could be a good summary that used to user understand the use case and gain insight from the experiments that we did.

Solution 1 is the first attempt to add C-API-S data structure into provenance relationship of Handle objects, using the Collection-Member relationships to represent the ‘HadMember’ provenance. The Handle System is the main service to present the nodes and relationships in Provenance relationship. C-API-S only provides the information about the ‘HadMember’ relationship. This solution is a simple data structure to solve the Complex data objects, but showing a incredible output on use case simulations. For use case with simple provenance relationship (type-to-type and all-to-all), it is inflexible to present the provenance relationship, worse than the Solution 2 which has a better strategy to present the relationships. However, when the use case is more complex as the real world case, benefiting from the Handle System, Solution 1 is the simple but fast way to represent the provenance relationship, maximizing the advantage of its flexible and quick inserting and querying.

Solution 2 is a advanced way to use Handle System and C-API-S, experimentally balancing the provenance information, moving partial provenance information from Handle System into C-API-S. There is a trade-off between number of requests and time costs in Solution 2 strategy. For the simple provenance relationship (collection-to-collection), collection-level provenance relationship, solution saves the time by ignoring the redundant accesses to the Handle System, proved in case 1 and case 2. However, when the member-level (type-to-type) provenance relationship is embedded, C-API-S consumes more time since querying performance of C-API-S is lower than Handle System. We can conclude that the solution 2 is a alternative approach to solve the situation that solution 1 cannot handle efficiently.

Solution 3 is a control group observing the C-API-S’s work on handling Provenance relationship individually. All experiment results show pure C-API-S is good at managing Provenance relationship, but is not efficient enough. Handle and C-API-S work together for mutual benefit, handling complex provenance relationship, and cost less time on representing the Complex Data object.

Also, for above three discussions, we add another conclusion about relationship communication cost of the C-API-S and Handle System. There is no doubt about the advantage of Handle System, but for designing the data structure in C-API-S, we encourage to use Collection and All Member operation since they both are efficient, saving time on representing and transferring the data object.

10 Future Work

This pilot evaluation is a first step to evaluate the strength and weakness of Handle System and C-API-S, observing how they represent the Provenance relationship efficiently. C-API-S has a lot of potentiality to manage and represent the Complex Data Object since its flexible Collection-Member data model. The next step of this research is going to figure out a clear understanding the Provenance relationship, and then maximizing the advantage of Handle System and C-API-S.

11 Acknowledgements

We thank the member of Research Data Alliance Collection Working Group for brilliant design and implementation, and thank the member of Research Data Alliance PID Kernel Information Working Group for insightful guidance.

References

[1] Rda recommendation on pid kernel information (version 1). [Online]. Available: https://doi.org/10.15497/RDA00031
[2] N. Paskin, “Digital object identifier (doi) system,” Encyclopedia of library and information sciences, vol. 3, pp. 1586–1592, 2010.
[3] J. Kunze, “Towards electronic persistence using ark identifiers,” in Proceedings of the 3rd ECDL Workshop on Web Archives, 2003.
[4] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao, “Efficient querying and maintenance of network provenance at internet-scale,” in Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD ’10. New York, NY, USA: ACM, 2010, pp. 615–626. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807234
[5] P. Chen, B. Plale, Y.-W. Cheah, D. Ghoshal, S. Jensen, and Y. Luo, “Visualization of network data provenance,” in High Performance Computing (HiPC), 2012 19th International Conference on. IEEE, 2012, pp. 1–9.
[6] I. Suriarachchi, Q. Zhou, and B. Plale, “Komada: A capture and visualization system for scientific data provenance,” Journal of Open Research Software, vol. 3, no. 1, 2015.
[7] J. Darmont, O. Boussaid, J. Ralaivao, and K. Aouiche, “An architecture framework for complex data warehouses,” CoRR, vol. abs/0707.1534, 2007. [Online]. Available: http://arxiv.org/abs/0707.1534
[8] W3c. [Online]. Available: https://www.w3.org/TR/prov-overview/
[9] S. Sun, L. Lannom, and B. Boesch, “Handle system overview,” United States, 2003.
[10] S. Sun, S. Reilly, and L. Lannom, “Handle system namespace and service definition,” United States, 2003.

[11] Data type registry. [Online]. Available: http://typeregistry.org/

[12] Collection api service. [Online]. Available: https://github.com/RDACollectionsWG/specification/blob/master/readme.md

[13] Research data collections wg. [Online]. Available: https://www.rd-alliance.org/groups/research-data-collections-wg.html