# Enabling User Collaboration in Science Gateways

## Apache Airavata Sharing Service

Supun Nakandala
Science Gateway Research Center
Indiana University
Bloomington, IN, USA
snakanda@iu.edu

Suresh Marru
Science Gateway Research Center
Indiana University
Bloomington, IN, USA
smarru@iu.edu

Marlon Piece
Science Gateway Research Center
Indiana University
Bloomington, IN, USA
marpierc@iu.edu

Sudhakar Pamidighantam
Science Gateway Research Center
Indiana University
Bloomington, IN, USA
pamidigs@iu.edu

Kenneth Yoshimoto
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA, USA
kenneth@sdsc.edu

Terri Schwartz
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA, USA
terri@sdsc.edu

Subhashini Sivagnanam
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA, USA
sivagnan@sdsc.edu

Amit Majumdar
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA, USA
majumdar@sdsc.edu

Mark A. Miller
San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA, USA
mmiller@sdsc.edu

## ABSTRACT

Managing workload for compute-intensive applications is a fundamental task to provide resilient, scalable and manageable infrastructure. In this paper, we discuss possible solutions to address most of the prominent distributed workload management problems for scientific applications. Scientific applications are mostly compute-expensive and require high end computing resources and may require a lot of time for execution. Scientist generally use these applications to run their analysis on large data sets which force these applications to deal with a lot of data. Our proposed system divides workload management logic into distributed component. This design is motivated from the microservice architecture to exploit its inherent support for scalability, availability and portability. Scientific applications have limited set of tasks in an application, our system identifies these tasks as a DAG (Directed acyclic graph) and process them based on user inputs. There are set of microservices which uses elegant communication infrastructure to fulfill end to end DAG execution. More importantly, our system is capable enough of absorbing any additions and modification to the tasks as they come on the fly.

## CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Information systems →** *Computing platforms*; • **Software and its engineering →** *Open source model*;

## KEYWORDS

Apache Airavata, SciGaP, CIPRES, NSG, SEAGrid, Collaboration, Science Gateways, Groups, Sharing

*PEARC17, New Orleans, LO USA*
2017. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

**ACM Reference format:**
Supun Nakandala, Suresh Marru, Marlon Piece, Sudhakar Pamidighantam, Kenneth Yoshimoto, Terri Schwartz, Subhashini Sivagnanam, Amit Majumdar, and Mark A. Miller. 2017. Enabling User Collaboration in Science Gateways. In *Proceedings of Practice & Experience in Advanced Research Computing, New Orleans, LO USA, July 2017 (PEARC17)*, 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

To be Written.

## 2 PROPOSED SOLUTION

First, we formulate our requirements in the form of a logical relationships between entities and come up with a logical model to cater to the requirements. We identify the concept of ?Sharing? as the cornerstone in developing a user collaboration system. Sharing is a ternary relationship between three abstract concepts, namely ?Actor?, ?Artifact? and ?Permission? (see Fig. 1). Our model is similar to other resource level security models such as the Sharepoint item level security model[5]. In our model, an actor can be a single user or a group of users. The concept of artifact represents any data entity in the gateway (e.g experiment, output file etc.) and the permission concept captures the level of access that each actor has on the artifact associated in the sharing relationship. Sample sharing rules written using this model are shown in Fig. 1. In our model, we assume all sharing rules are explicitly defined (i.e no default behavior) and to keep the model simple, we also do not support sharing rules defined as negations, such as DISALLOW (Group1 has READ to Entity1). When an Artifact is created by the default the owner of the Artifact will be assigned with the OWNER permission which grants the global level permission on that particular Artifact. Next we look into each of these concepts and the different operations the model needs to support.

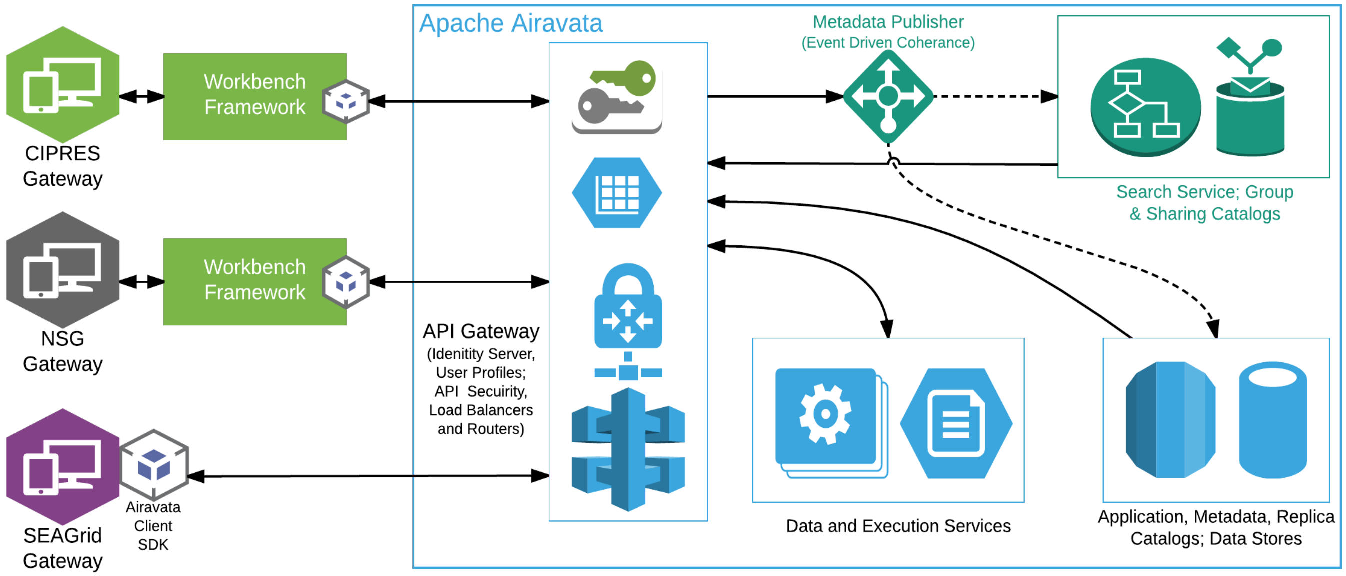


**Figure 1: High level architectural overview. SEAGrid Gateway is built over Apache Airavata for all data and execution capabilities; CIPRES and NSG Gateways are built over Workbench Framework and are exploring use of Apache Airavata for Data Sharing capabilities. The paper focuses on Sharing Service (green color box) and existing literature on Airavata discuss other capabilities (blue colored boxes).**

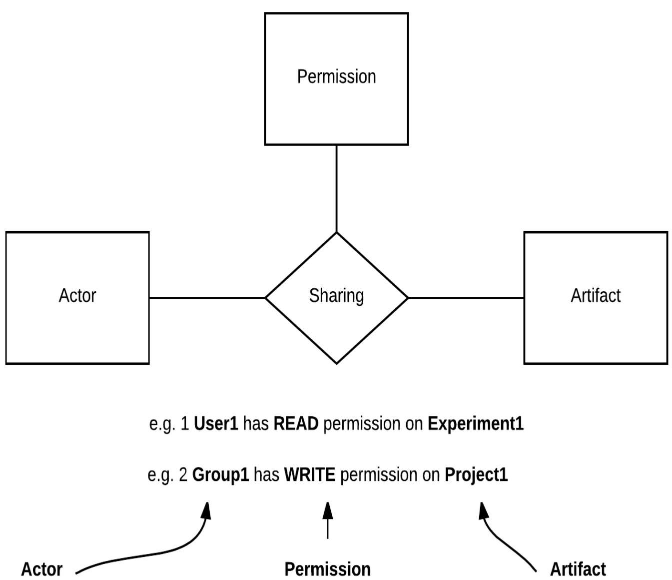


**Figure 2: Relationship between the Actor, Permission, Artifact and Sharing concepts.**

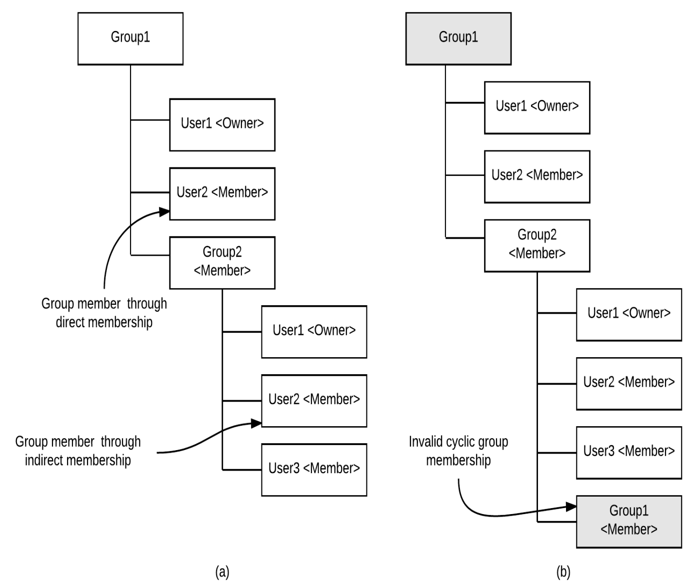


**Figure 3: (a) Composition of a valid group with nested group memberships. All members of Group 2 are also members of Group 1. (b) Composition of an invalid group because of a cyclic group membership.**

Actor: As mentioned earlier, an Actor who participates in a sharing relationship can be either a single user or a group of users. A group will be owned by the creator of the group and can have other users or groups as child members. Child member groups can in turn have other groups, which are owned by the same owner, as child members, but cannot have cyclic group memberships. A particular user can be a member of a group via direct membership or via indirect membership (i.e. by being a member of another nested group or both). These concepts are shown in more detail in Fig. 2.

Artifact: An artifact can be any type of data item, either real or virtual, that exists in the gateway environment. From a conventional gateway point of view, items like user projects, jobs and input/output files can be considered as artifacts. In a more generic sense, any item that needs to be shared among users or needs to be controlled with different access requirements for users falls into the category of an Artifact. An artifact can have a hierarchical structure (i.e parent-child relationships). This hierarchy becomes important when a user wants to share items at different granularities. For example when sharing a project, a user may want to share all the nested child experiments and input/output files implicitly, or share only the project content without child entities. Similar to Actors, Artifacts cannot have cyclic dependencies in their parent-child hierarchies.

Permission: The notion of permission captures the different access types that an Actor can have on an Artifact in a sharing relationship. For example, User1 may have READ permission on Experiment1, which will only grant access to view things in Experiment1 where as Group1 may have WRITE permission on Experiment1 which would allow a Group1 member to write/update the content in Experiment1. Sometimes it is possible that one permission may subsume the access rights defined by another permission. For example the OWNER permission (granted to the owner of an artifact) subsumes the access rights enforced by READ and WRITE permissions. A particular user may have READ permission either because the item is explicitly shared with READ permission or the user has another permission (e.g. OWNER) which subsumes the access rights of the READ permission. These kinds of relationships are captured in the form of a permission hierarchy. Similar to Actors and Artifacts these hierarchies also cannot have cyclic parent child dependencies.

Operations Supported: In addition to supporting the basic CRUD operations on the entities(Actor, Artifact and Permission) the model should support three different types of operations. They are: ? Sharing/Revoking - Explicitly sharing an Artifact with an Actor with some Permission and revoking previous sharings. The sharing can be done at different granularities. Such as sharing a parent Artifact including all its child Artifacts or sharing only the parent and not the children. ? Check Permission - Checking whether a specific user has a specific permission to a specific artifact. ? Get all accessible Actors - Retrieving the list of all actors who has the specified permission to particular. ? Search/Browsing - Searching across all accessible Artifacts for a specific user with some Artifact search criteria. Browsing can be considered as a special case of search with an empty search criteria on Artifact fields.

## 3 IMPLEMENTATION OF THE PROPOSED SOLUTION

For the implementation we developed a system backed by a relational database engine where entities and relationships are mapped into relational tables. To keep the implementation simple we treat individual Users as a special type of Group which can have only one member. To elaborate more on how we support the different operations of our model and to make the discussion lucid, we formulate a hypothetical scenario. In our scenario we have three users, three permission types hierarchy (OWNER, READ and WRITE), the

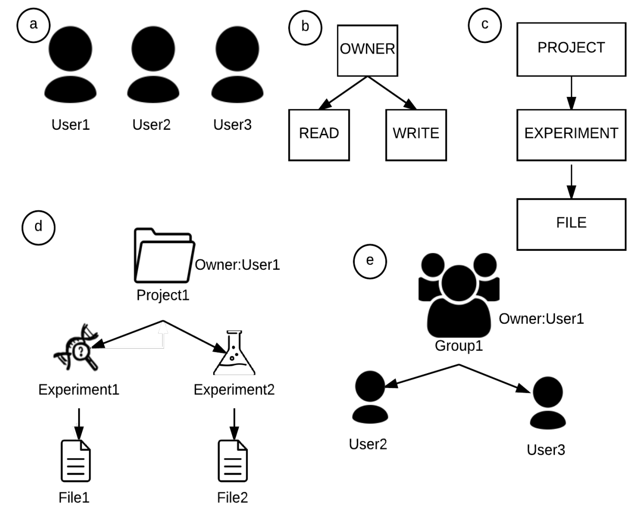


**Figure 4: Hypothetical user collaboration scenario. (a) users in the scenario, (b) Permissions hierarchy, (c) Artifact types hierarchy, (d) Artifact hierarchy owned by User1 and (e) Group hierarchy created by User1.**

artifact types hierarchy (PROJECT, EXPERIMENT, and FILE). User1 owns one project (Project1) in which there are two experiments (Experiment 1 and 2) each with one file (File 1 and 2). User1 also has created two groups (Group 1 and 2) in which one is a nested subgroup in the other.

Sharing and Revoking: Initially only User1 has access to Project1 and all other nested artifacts (through the default assignment of the OWNER permission). Let us assume that User1 wants to give READ permission to Project1 for User2. There are several ways to achieve this. The most basic approach would be to explicitly share Project1 with User2 with READ permission. This operation corresponds to creating an entry in Sharing which can be represented as

However this will allow User2 to READ Project1 and not any of its nested artifacts. If the User2 also needs to have READ permission to all nested artifacts of Project1 we need to define sharing rules for all the nested artifacts rooted at Project1, either explicitly or implicitly. The implicit way to handle this is to add an additional field in the Sharing rule to capture this information. The explicit way to handle this is to traverse through the Artifact subtree and create sharing records for each nested Artifact. The two approaches can be represented as follows:.

There are tradeoffs in choosing between these two approaches. The implicit approach has constant time complexity at sharing time but requires complex and computationally expensive retrieval operations at inference time (i.e Check permission and Search operations). On the other hand, the explicit approach has O(n) time complexity (n is the number of artifacts in the subtree) and requires constant time complexity at inference time. In a practical setting, most operations the sharing model should support will be inference type operations. Hence in our model we have decided to use the explicit approach. One might question why we need to maintain the CASCADE field in the Sharing rule when using explicit approach where we create sharing rules for all child Artifacts. To explain this requirement, consider the following case. Assume that User1 has shared Project1 and all its child Artifacts with User1 with READ permission. Later User1 creates a new Experiment (Experiment3) which is rooted under Project1. According to our model User2 should have READ access to the new Experiment3, even though it is created after the entire project was shared. To achieve this at the Experiment3 creation time, it should inherit all the Sharing rules from its parent which are cascadable. This to determine whether a sharing rule is cascadable we need to maintain the CASCADE field in the Sharing rule.

The model also provides the capability to revoke already defined sharing rules. This operation corresponds to the deletion of the specific Sharing rule. But some complications may arise in the case of cascading sharing rules due to multiple inheritance yielding the same Permission. For example, consider the following case. Assume Experiment1 and its child artifacts are shared with User2 with READ permission. This allows User2 to READ the File1. The corresponding entry in the Sharing can be written as

Also assume that later, User1 also decides to Share the Project1 and all its child artifacts with User2 with READ permission. This Sharing rule will also enable User2 to READ File1. But note that the corresponding Sharing rule entry for Experiment1 and File1 will be same as in the previous case. This will either duplicate the Sharing rule or would not make any changes, based on the implementation. Later if User1 wants to revoke the Sharing defined on Experiment1, all the Sharing rules that were created as a result of that Sharing have to be deleted as per the following operation.

This will end up having only one Sharing rule allowing User2 to READ and Project1 and nothing else. Hence it is important to capture the inheriting parent Artifact when recording cascading Sharing rules and at the revocation time deleting only the ones that correspond to the specific revoking parent Artifact. The modified Sharing operation on Project1 will now look as follows.

Going back to our first scenario, granting User2 with READ permission on Project1, we could have achieved this in two other ways. One approach would be to use the Permission type hierarchy and Share with a Super permission of READ (e.g). An actor who has OWNER Permission on an Artifact will also have READ Permission. The other approach would be Share the Artifact with an Actor that already contains the User2. For example if Project1 is Shared with Group1 with READ Permission, then User2 will also have READ Permission on Project1 as a result of being a member of Group1. Similar to handling nested hierarchies in Artifacts, the nested hierarchies in Permission types and Groups can be handled either implicitly or explicitly. The implicit approach will record only the original Sharing rule and will defer the heavy lifting to the inference time. On the other hand, in the explicit approach addition records will be created to each nested element. For example when Sharing with OWNER permission additional records will be created to READ and WRITE permission. However we found that most of the Permission Type and Group hierarchies in our use cases are short and flat and always will be of cascade type (e.g. when sharing with a group it is always implied that it is sharing with all the nested members). Therefore the overhead of using the implicit approach is not significant. Hence we use the implicit approach to handle Permission type and Group hierarchies.

Check Permissions: In our model since we put most of the heavy lifting on the sharing and revoking operations, inference operations become straightforward and less costly. The Check Permissions operation tries to find out whether a specific Actor has a specific Permission to a specific Artifact. For example consider that we want find whether User2 has READ Permission to Project1. In our model, which implicitly captures nested aspects in Permissions and Groups, we need to first find all the Permissions which subsume the READ permission and all the groups that User2 is a member of. After finding those, this operation can be easily executed as follows.

Get all Accessible Actors: Search/Browse: In the search operation the objective is to retrieve a list of Artifacts which are accessible to a specific actor based on some permission criteria and which conforms to a particular search criteria on Artifact properties. As explained earlier, Browse is special case of search with empty Artifact filter criteria. As a practical requirement, when dealing with large return lists, it is also important to handle pagination. As the objective of this operation is to retrieve the list of Artifacts we do a relation join on Artifact and Sharing entities and then apply the filter criteria. In the current implementation, the Artifact filter criteria can be applied on Artifact name, description, owner, parent artifact (if exists), created and updated times and the full text field. Some of the example search/browse queries will be as follows: ? Browse without Artifact type constraint: e.g. Select Artifacts where User2 has READ access. ? Browse with Artifact type constraint: e.g. Select Artifacts of type EXPERIMENT where User2 has READ access. ? Search with Artifact filter criteria: e.g. Select first 10 Artifacts of type EXPERIMENT where User2 has READ access and the Artifact name contains ?Ethylbenzene? and Artifact created during last week. The last search operation listed above can achieved as follows:

## 4 INTEGRATION WITH SEAGRID

SEAGrid provides a traditionally secure data model where all the data has been considered to be proprietary and data sharing occurred only outside the SEAGrid application. However, recently users have requested the data be shareable among collaborators to improve organization, post processing and collaborative discovery. The discovery and sharing of the data go hand in hand as the simulation data accumulates over time and data management in terms of discovery for specific goals or to reuse in specific ways becomes critical. Supporting data discovery by individual users and collaborative processing of data requires both metadata generation and data indexing. The original SEAGrid infrastructure provided a way to define key value pairs so users can manually add to a metadata catalog for each experiment they set up, along with a search engine to discover the tagged data and products. However, users did not adopt this system, as it required a disciplined addition of metadata tags. Recently we eliminated the manual tagging operations by implementing an automated parsing and indexing system. This system generates metadata tags for output of some of the applications used in the SEAGrid gateway [ref]. For an application that generated post-processed visualization data products we also implemented automatic metadata-enhanced archiving of results into the SeedMe.org archive [ref] for the Vortex-Shedding Gateway [ref]. The parsing infrastructure created for the SEAGrid gateway provides access and discovery of any data shared with the user, as it becomes part of the searchable domain. Additional sharing infrastructure requires us to define what is shared and with whom and a way to collaboratively process data. Toward this end we have prototyped integrating Jupyter notebooks in a GSOC project that can be integrated into Apache Airavata gateway infrastructures to interact with data products with ready analysis functions available in python [ref]. The users have benefitted from the job history organization already available in the portal but were unable to exploit the vast amount of past data from the community as the infrastructure to securely share the data. A generic way to publish data to community organizations such as Figshare [ref] or SeedMe.org have just been implemented but more controlled sharing for collaborative and cooperative post-processing need additional infrastructure to define collaborating individual or group and mode of sharing and the timelines. Sharing of post-processing functions/applications along with the data and/or provenance details would also be beneficial for reproducing data products as well additional enhancement to the analysis if needed.

# 5 INTEGRATION WITH THE CIPRES WORKBENCH FRAMEWORK

The CIPRES Workbench Framework [refs] was created with the assumption that each authenticated user was working independently of all others, which is consistent with the work paradigm of biological science when the software was designed. The software at present does not support collaboration or data sharing between individual accounts. However, recent surveys show that users of the Neuroscience Gateway (NSG) and the CIPRES Science Gateway, two highly accessed gateways built using the CIPRES Workbench framework, have a strong interest in sharing data within their workgroups, and with the public as well [REF-Majumdar; Miller, unpublished data]. Both user groups indicate that data access and data sharing presents a significant challenge, and that users would appreciate a solution that can be managed from within the Gateway interface. Two specific use cases highlighting the data sharing needs of typical Workbench Framework users are listed below: A geographically distributed, highly interdisciplinary neuroscience research group is interested in collaboration that combines empirical results from MEG experiments with detailed computational models of neocortical circuitry to infer the mechanisms responsible for generating cortical rhythmic activity. Experimental data is collected from geographically distributed collaborators' institutions, yet jointly all of them are developing a large-scale model of thalamocortical circuitry with detailed neurons and synaptic architectures and plan to run on supercomputers. Model predictions will be validated and informed with simultaneous thalamic and cortical microelectrode recording data (that needs to be shared) obtained in the collaborators' lab in other institutions in US and France.

EEGLAB [REF-EEGLab] is a widely used software for processing experimental eletrophysiological data by thousands of cognitive neuroscientists worldwide. Researchers are now interested in meta-analysis of source-resolved EEG measures across studies. This will allow research results of individual researchers/labs to be analyzed/compared (meta-analysis) to gain deeper understanding of brain functions at a higher level. To be able to do this researchers will need a mechanism, within NSG, where they can make their data shareable such as by tagging results that they want to share with other researchers or make it public. This will require different level of sharing capabilities. This will then allow other researchers to perform meta-analysis of research results at a higher level using computing power provided by NSG.

Based on user feedback, and the use cases above, tools for sharing data between accounts are a priority feature for CIPRES Workbench users. We are integrating the data sharing service with the CIPRES Workbench framework. Our approach will take a somewhat different strategy from the SEAGrid integration described above. A schematic is shown in Figure 4. For the Workbench Framework, a separate group management server will allow users to perform group and user management. This will allow full access to all functions in the SciGap sharing service without any User Interface development. CIPRES Workbench will retain an internal representation of Experiment configuration, so the shared Entities will be project folders. Within the Workbench code, project folder identifiers will be used to determine user access to contained Data and Experiments (Tasks in workbench terminology). Integration with the Experiment Catalog will not be required.

# 6 RELATED WORK

Internet2 Grouper software[6] is an access management system which is widely used in universities that can support most of the required features in our proposed solution. Grouper provides comprehensive and efficient group management functionalities including rule and time based group membership definitions and ability to define groups based on complex group math operations (unions, intersections and differences) which are generally beyond the simple group management requirements in gateways. Features in Grouper such roles, attributes, and permissions can be used to implement our sharing model. If implemented using Grouper, the system can easily support the sharing(revoking) of Artifacts, check permission, get all accessible users operations. However the lack of flexibility in the Grouper?s search/browse APIs hinders the system in efficiently fulfilling the search/browse operations required by the Gateways users. Search/Browse operations are an important aspect for Gateways users and hence we decide to implement our own implementation with flexible search/browse operations. SeedMe[7] is a cyberinfrastructure platform that enables seamless sharing and visualization of computer simulation outputs which runs on HPC infrastructures through a web based tool. It enables users to conveniently view and access simulation outputs. Users can create collections of data items and share publicly or with a specified group of users. In the current implementation it has only one permission level which grants permission to shared users to view and comment on the collection. Also it lacks the ability to define reusable user groups and requires explicitly defining the list of users every time when sharing a collection.

# 7 CONCLUSIONS