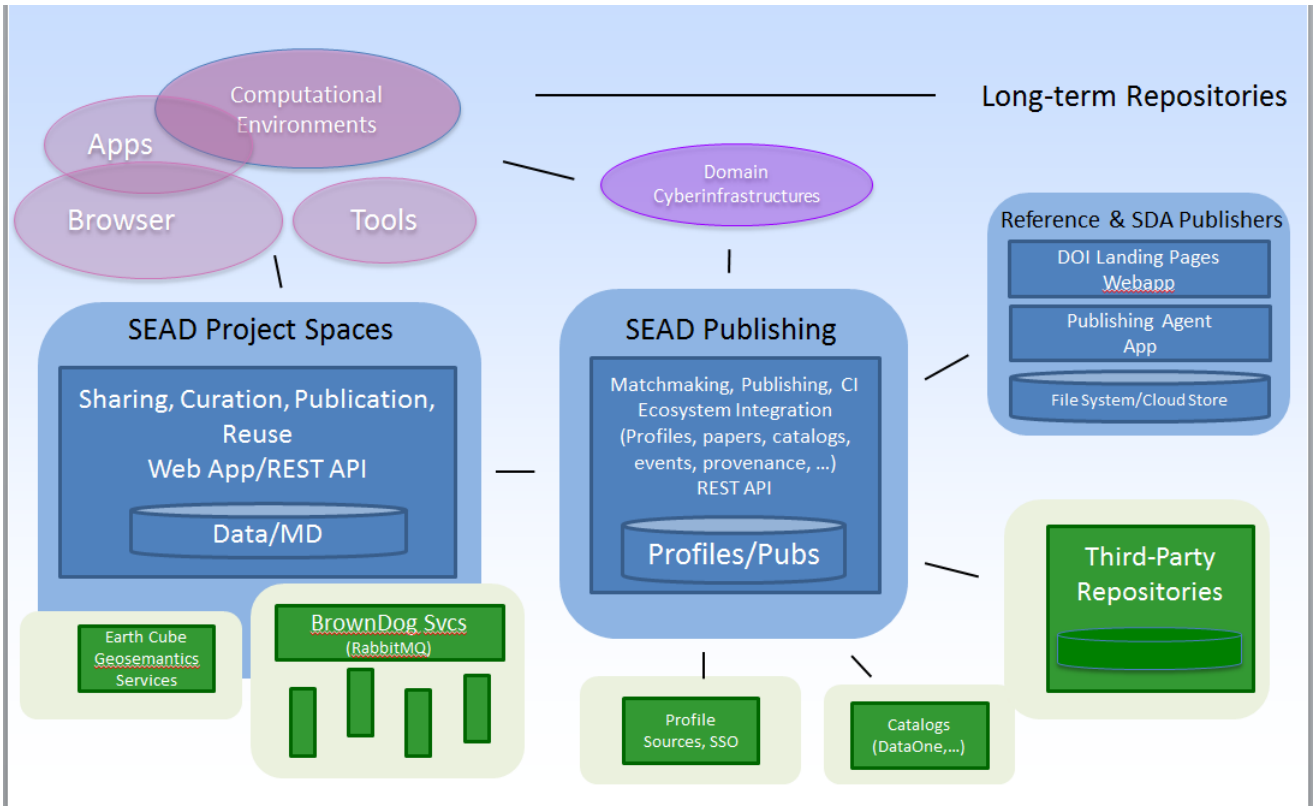
SEAD Reference Publisher



The SEAD model for publishing involves (more information available on other wiki pages and SEAD publications/presentations):

- Secure active Project Spaces in which groups work, during their research projects, building collections of data files, annotations, and metadata (through a web interface or via researcher's software writing directly to their project space via the SEAD API),
- Publishing Services that cache, analyze, and manage the submission of publication requests from researchers to repositories partnering with SEAD. These services use information about repositories and their policies (submitted by the repository), about people and their affiliations (harvested from sources such as ORCID), and the contents of publication requests (submitted by researchers from their project spaces or third-party infrastructure), to 'match' requests with compatible repositories on the basis of a configurable set of rules that may involve limits on total size, maximum file size, file types, affiliations of authors, and other metadata and statistics. In SEAD's 2.0 version, researchers can review the results of this assessment and decide whether to adjust their publication requests (adding required metadata, removing unacceptable files, etc.) and to then submit to a repository of their choice. SEAD's publishing services then make such requests available for repositories to discover and retrieve, and then tracks the status of the repository's processing of a request through to the final completion and assignment of a persistent identifier (e.g. DOI).
- Repositories partnering with SEAD to acquire and preserve data publications that meet their institutional interests. Repositories may range from institutional repositories with rich services (e.g. based on DSpace or Fedora4) to repositories with fewer features but lower costs and/or higher scalability. SEAD provides a RESTful API (see the SEAD 2.0 Publication API Walkthrough:) that repositories can use to interact with SEAD and retrieve data publication requests. Repositories working with SEAD had demonstrated connections via this API to institutional repositories. SEAD has also worked to develop software tha can efficiently and scalably package data publication for storage and retrieval of publications on large file systems and cloud storage. All of this software is avialable for use and/or extension and modification by respositories interested in connecting to SEAD.

The SEAD Reference Publisher

The SEAD Reference Publisher is a lightweight web application that can publish and provide access to data publications originating in SEAD. It consists of two interacting components that manage 1) The SEAD publication process from retrieval of a request from SEAD's publishing services to packaging, storage, and submission of a persistent identifier back to SEAD, and 2) providing access to the stored data through a landing page associated with the persistent identifier minted for the data. The reference publisher serves several purposes:

- It supports publication of the larges packages SEAD has had to date (135K files, >160GB, 40+GB individual file size) and allows testing of SEAD's Spaces and publishing services at this scale
- It demonstrates good (minimalist) handling of SEAD publication requests in terms of minting DOIs with discovery metadata, packaging submissions in a standards-based format, and supporting ongoing access to the data and metadata
- It provides integrity checks of metadata submitted with the request (verifying the output of publications sources such as SEAD's v 1.5 and 2.0 spaces, and verifying that the contents of the final data package are consistent with the file lengths, hash values, total file/size counts, and hierarchical structure defined in the metadata).
- It provides the core publication/packaging capability used within the Indiana University SEAD Cloud repository, which offers SEAD users a place to publish data (current policies limit submissions to a maximum size of 1 GB)
- It provides an out-of-the-box capability that can be directly deployed or used as a starting point for institutions wishing to partner with SEAD (initial deployments are being pursued at the University of Michigan in partnership with UM's Advanced Research Computing (ARC) organization and as part of a pilot effort with the National Data Service).

What it does

The SEAD Reference Publisher (SRP) is packaged as a single war file that generates the landing page and download links for published data. This war file also includes classes that can be run locally to publish data.

Publishing:

As currently implemented, the SRP is configured with the location of SEAD's publishing services, and, to get started, the operator must submit a json-ld profile for their repository instance to that service endpoint (using its restful API). Once that is done, researchers can send publication requests to this repository and the operator can monitor SEAD's services to see when new requests appear. For the SRP, these steps are manual, but the IU SEAD Cloud has demonstrated how this discovery phase can be automated and a repository can run a simple cron job to automatically pick up new requests. Once a request is discovered, the SRP can be run at the command line, given the Identifier for the request, and it will being retrieving the information required to publish the data:

The SRP initially requests the publication request document which provides a summary of the metadata (title, creator(s), abstract, publishing project id/name...) and overall statistics (size, # files, file types included, ..) and a link to the full metadata document.This information is intended to be sufficient for a number of basic policy-based decisions that might influence whether a repository is willing to accept and process the request. The SRP currently accepts any request (the operator is assumed to have made the decision to accept the request).

The SRP uses the link to the full metadata document to retrieve it. SEAD's metadata documents conform to the proposed JSON-LD serialization of the OAI-ORE standard for describing "Aggregations" and their "Aggregated Resources". Publications in SEAD are represented as a single "Aggregation" that includes a flat list of "Aggregated Resources". These are currently either sub-folders/sub-collections or individual files, with each such resource having it's own metadata. SEAD uses the Dublin Core "hasPart" relationship to describe the intended hierarchical structure, i.e. the Aggregation has an array of "hasPart" entries that are the identifiers of "Aggregated Resources" included in the document that represent the top-level of folders/files in the publication request. Folder resources in turn have a "hasPart" array that includes their children. SEAD will also send other types of relationships that are represented in the metadata, but these are not used by SRP to structure the final publication package.

The metadata document includes links for all files ("Aggregated Resources" that have an associated byte stream) that point to the content in the originating project space. As the SRP parses the metadata document to identify resources and learn the hierarchical structure of the package, it records these links. Once parsing is complete, the SRP calls an EZID library to mint a DataCite Digital Object Idetifier (DOI) for the data, and it then begins to generate the files and directory structure required by the BagIT format developed by the US Library of Congress and begins to stream content into a single zip file for the publication. When minting the DOI, the SRP provides basic metadata about the title, creator(s), abstract, publisher, publication date, and related information to DataCite.

The SRP uses the Apache concurrent zip library that allows incoming content to be directly compressed and stored (using one intermediate temporary file per thread) into the final zip file. The SRP records all necessary metadata files (e.g. the manifest list required by BagIT, the list of hash values associated with each data entry, the full metadata map, and a file linking the identifiers used in the metadata document to the paths to data files within the BagIT structure) and then beings to stream data files into the zip. By default, SRP uses one thread per cpu core, but it can be configured to use more to enable more parallelism in the file transfers from the originating Project Space.

Upon completion of the zipped Bag, the SRP opens it for reading and tests the integrity of the contents (that file lengths and hashes match those stored in the Bag, that the total number of files, and total number of bytes match those provided in the original publication request).

As a final step, the SRP determines whether the request have succeeded or failed and sends a final status message to SEAD's publication services. A failure message includes text desribing the nature of the failure, while a success message includes the minted DOI, which SEAD's publishing services then transmit back to the originating project space.

Access:

The primary means of access to published data is via the minted DOI. SEAD displays the DOI for the published data as part of the "Published Data" page within researchers' Project Space and submits it to the DataOne metadata catalog where it can be discovered via their search interface. Researchers may also include the DOI for their data in papers or may include a reference to a paper in their data which may be harvested by publishers to create a reverse link to the data via its DOI. All of these mechanisms rely on use of the DOI to redirect to a landing page through which the data and metadata can be accessed. A DOI can be represented as a URL which preprends the name of a resolver web service that, when submitted by a browser, will redirect the user to the landing page representing the identified data. The SRP web application generates this landing page dynamically using several related techonologies and with some minimal caching of information:

The URL the DOI resolves to includes an identifier that allows the SRP web app to identify the specific zip file on disk containing the requested data publication. (These files are arranged in a hash-based directory tree that distributes the files across many subdirectories to make acess more efficient). The SRP reads the information required to generate the landing page directly from the metadata document stored within the zip file. The SRP relies on the Apache zip library, which creates an internal index within the zip file to allow efficient 'random-access' to any file within the zip, to read only the bytes for the metadata file.

The first time the SRP reads a specific metadata file, it will create two files on disk - one including just the metadata for the publication as a whole, and one that indexes the offsets within the metadata file of the metadata for specific resources (files or folders in the publication). These two steps improve the performance on subsequent accesses. They are usually minimal in size (e.g. <<1% of the publicaiton size) and can be regenerated if deleted.

The landing page itself is an HTML page that calls a javascript library that makes RESTful calls to the SRP web application to retrieve the metadata necessary to populate the page. The javascript initially retrieves the metadata for the overall publication and displays it as a series of fields on the page. It then retrieves information about the contents of the publication, starting with the top-level files and folders, An open source library is used to display these entries as a hierarchical tree of files and folders, with folders initially closed/collapsed. As a viewer clicks to expand a specific folder, the SRP requests the direct contents of that folder (e.g. it's child files and folders) and displays those. This process makes it possible to display any folder's contents almost immediately as the viewer clicks in the page.

The page displays links to the overall publication zip file (which can be very large), the metadata file, and, within the table of contents, links to each individual file in the publication. Clicking in any of these links results in a request to the SRP that serves the requested file back to the browser for download. (Despite the fact that the SRP stores all of these files within one zip, the efficiency of the APache zip library in retrieiving only the bytes for the specified file, and the fact that individual files can be much smaller than the overall collection, can make retrieval of the desired subset of files much faster than downloading and opening the overall zip file.)

The SRP does not currently provide a home page providing a list of published data sets or any search interface. The IU SEAD Cloud has demonstrated one way this may be accomplished (using an alternate landing page mechanism while sharing most of the publication library with the SRP). The SRP also does not track download statistics, but the means to optionally provide such information to Google Analytics is being added.

Deployment

At the University of Michigan, the SRP is being tested using a virtual machine being provided by Advanced Research Computing. The machine has been configured with CentOS 7 with 2-cores and 4 GB RAM with a 50 GB system disk and 2 TB of data publication storage. The machine is firewalled to allow public access on port 80. Yum was used to install tomcat, nginx, unzip, and audit2allow. Niginx was configured to forward port 80 requests to Tomcat at port 8080, and audit2allow was used to allow nginx to manage and forward port 80. The SRP was uploaded as a single war file that was unzipped and deployed to Tomcat. It was configured to use 8 threads, mint "test" DOIs (which only work for two weeks, but require no EZID credentials), to store data on the large 2 TB partition, and to use SEAD's test publication services. Log4j was configured to provide a daily rotating log for the SRP. A simple publication script was created that runs as "tomcat" and uses the class files and libraries within the webapp to publish data. A profile for the UM-ARC-TS repository has been added to SEAD's test publication services (http://seadva-test.d2i.indiana.edu/sead-c3pr/api/repositories/UM-ARC-TS).

Performance

This SRP instance has now been used to publish data packages ranging from 3 files/1.7MB to one including 135K files, 169GB total.

The following examples are from the National Center for Earth Surface Dynamics and were published form the NCED Space SEAD runs on a cluster at NCSA (see the Published Data page at https://nced.ncsa.illinois.edu/acr/#discovery): Given that they use test DOIs, these links will expire in ~2 weeks, but we anticipate publishing further tests and/or long-term publications that would be listed on the NCED Space Published Data page.

3 files, 1.77 MB à 557KBzipped, http://dx.doi.org/10.5072/FK22N54B43

11625 files , 22GB à20GBzipped http://dx.doi.org/10.5072/FK2NG4NK6S

29431 files, 31.2GB à 30.8GBzipped http://dx.doi.org/10.5072/FK2930TN6Q

135K files, 169GB à 113GBzipped , http://dx.doi.org/10.5072/FK2XK89H7D

Several issues have been addressed to successfully publish the larger collections:

SEAD Publication services required a larger Java Heap size to successfully serve the largest metadata files (which were >150MB)

The SEAD Publication services check of data file URLs (to verify that they are accessible) was modified to handle 0 byte files, intermittent failures (we observed a few timeouts when retrieving files that we believe were caused by heavy load (unrelated to the publication request) on the originating Project Space), and to improve performance.

The SRP implemented a URL retry to address the possibility of intermittent GET timeouts. (After implementing, we saw no need for retries in publishing >160K files).

The SRP RESTful web services and landing page were redesigned to support incremental, asynchronous loading of content. (The original design, which loaded the full json metadata file from the server, caused browser memory issues for sizes > ~25 MB.) Further performance improvements are being explored.

To support the largest publications, the publication script redirected Java to use the large 2TB permanent storage space for temporary files (since the 169GB package size was > 50GB system storage).

The Java Heap size also had to be increased to 4GB (2GB was not enough). Heap size does not have to be increased to support serving landing pages.

Publishing Performance:

On the UM-ARC-TS VM, retrieving metadata from Indiana University and content from a Project Space on SEAD's cluster at NCSA, we observed the following:

For a publication with 11625 files, 22GB:

<10 minutes to generate request and package metadata within the originating Space (i.e. pulling metadata via queries from SEAD's 1.5 spaces database (RDF middleware over MySQL))

~40 minutes for SEAD C3PR at Indiana to sequentially test all file URLs (probably avoidable/optimizable)

~30 minutes to retrieve and process full package @ ARC-TS (from retrieval of the request to completion of the creation and storage of the ORE /BagIT/zip file)

~20 minutes to sequentially test SHA1 hashes of all files (a full scan of the created

From the publication of the 135K file/169GB publication, processing @ARC is ~linear with total size (~250 minutes for 113GB, versus 50 minutes for 20GB)

Landing Page Performance:

Initial loading, downloads, and incremental updates of the folder hierarchy as viewers browse the table of contents appear to have reasonable performance - <1 sec to start transfers, <1 s to show a folder with ~10 entries, slowing to a few seconds for folders with hundreds to 1K+ entries. The primary bottleneck is in loading the initial information for the top-level of the table of contents which, for historical reasons, involves a synchronous transfer and multiple RESTful calls (one per top-level entry). Converting this to a single asynchronous ajax call should make performance reasonable even for the largest collections we have to date.

Discussion:

Overall, serving content appears to be lightweight – minimal CPU load and minimal memory. Clearly higher than serving static web pages but the current VM should easily handle simultaneous users.

Publishing and packaging is CPU and memory intensive. The largest package (135K files) has a ~158MB metadata file and we currently download and parse that as one unit. That required giving Java a 4GB heap size. Once that files is parsed, we set up 135K HTTP transfers (one java object in memory per entry). On the 2 cores, it looked like 8 threads was ~optimal and the cpu load reported by 'top' stayed at 150-200%. To handle the largest packages, I moved the /tmp storage of the parallel retrieval (8 files – one per thread storing the compressed results streaming in) on the final storage (/sead_data). The bandwidth from NCSA to disk during the write phase was <=20MB/s (which includes the time for copying the 8 tmp files into one final zip on the same file system, etc.). Since write is one-time and we're primarily looking at smaller packages, this is probably fine, but it might be worth some discussion to see where the bottleneck(s) are and if there are easy ones to remove either at UM or NCSA.

Further work:

Some minimal additional work to remove the synchronous RESTful calls in the landing page should be all that's needed to provide reasonable performance on the existing VM. Beyond that, there are a number of areas in management/monitoring where we may want to adjust the machine configuration and/or create some tools:

The publication script could be automated (e.g. cron job) and we could leverage the policy-enforcement additions made at Indiana to only process compliant requests and to alert an operator (e.g. via email) when a non-compliant request is submitted and/or to allow out-of-band communication with users regarding publications).

Test DOIs expire in two weeks, so it is useful to remove test publications older than 2 weeks, which requires removal of the zip file (and the two associated cache files), This could be manual or on a timer. If we decide to support test and 'real' publications using the same instance, this script would need to read the DOI information in the metadata to assure that only data with test DOIs are deleted.

The SRP supports Google Analytics and should soon provide information both about views of the landing page as well as any subsequent downloads of the individual files or zipped publication.We may want to configure this to send information to SEAD or ARC.

If we want an overall home landing page and/or search capability across publications, we should be able to adapt code developed at Indiana. (Conversely, some of the performance improvements made in testing at UM may be usable at Indiana).