Scientific Data Management System (SDMS) - Statement of Work

Oak Ridge National Laboratory

Oak Ridge Leadership Computing Facility

Advanced Data and Workflows Group

November, 2017

# Introduction

There are currently many disjoint data management technologies available to scientific researchers at leadership-class computational facilities. There is a proliferation of both commercial and ad hoc systems that provide specific capabilities, such as data access and movement, indexing, dissemination, publication, and curation. Most existing systems are oriented toward enterprise (business) environments and do not address the complexities of HPC environments. The lack of integration, consistency, and HPC suitability among these technologies places a significant burden on scientific researchers and groups who must adopt, learn, configure, then follow complex and/or tedious manual procedures to utilize a piecemeal data pipeline. The result can be wasted time, confusion, and even errors related to properly identifying, locating, provisioning, and disseminating large scientific datasets.

The SDMS is envisioned as a system of integrated technologies that will provide a simpler, consistent, and holistic solution for the common data management needs of scientific researchers and users of HPC systems at ORNL. The SDMS will bridge the gap between existing enterprise-oriented solutions and the complex distributed environments associated with high-performance computing. The SDMS will provide the ability to uniquely identify, manage, and provision data entities from their point of origin throughout their entire lifetime for any data processing pipeline utilizing any or all supported compute environments.

Within the SDMS, data entities will be unique, atomic, and stored and indexed centrally and will have system-wide (“global”) visibility such that they can be uniformly accessed from any local compute environment. While central storage is straight forward to implement, *every* data access to central storage incurs a data transfer penalty (sometimes very large). For certain applications, such as workflows, this approach is an anti-pattern. The SDMS will utilize automatic local caching to eliminate the access penalty for repeated data accesses. User-level computational processes (i.e. jobs and workflows) will be able to easily utilize the SDMS to generate, retrieve, or update managed data entities without concern for the complexity or efficiency of access patterns.

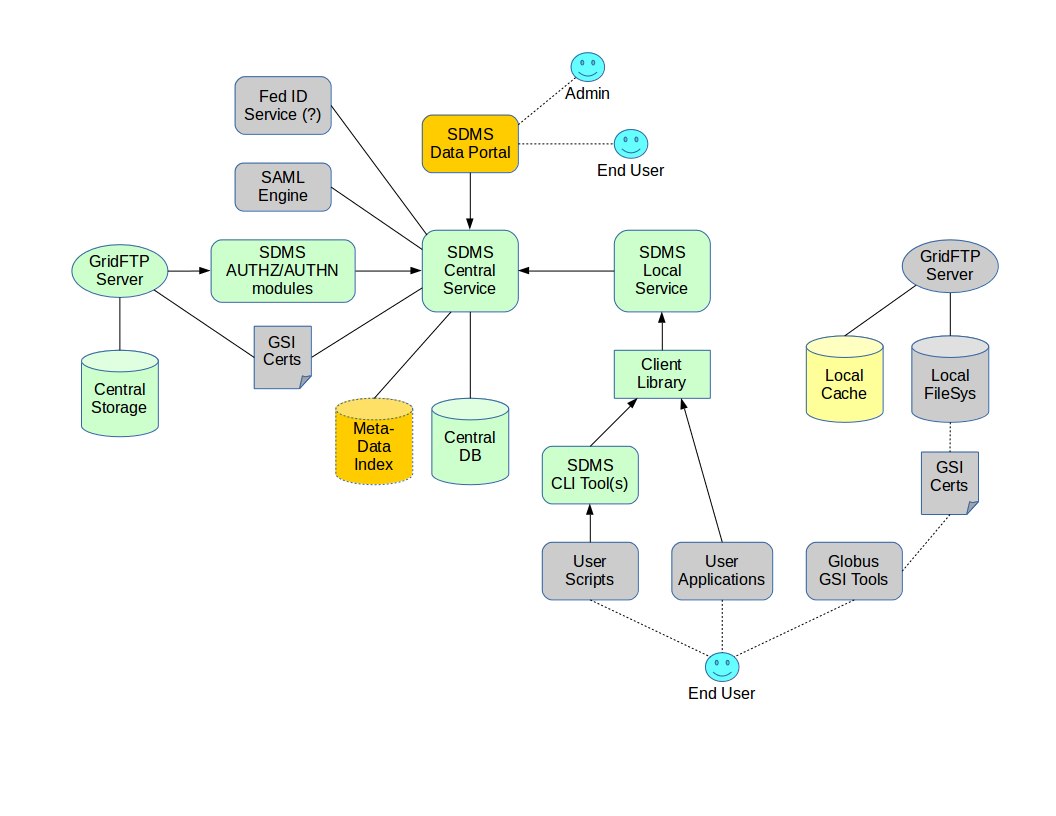
The SDMS will provide the following features and capabilities:

* System-wide, unique data identifiers
* User- and group-specific identifier aliasing
* Centralized data storage
* High speed data transfers (GridFTP)
* Automatic local caching
* User- and group-level access control
* Data version control
* Subscriber data notifications (can be use for data pipeline)
* Metadata indexing
* Python client library
* Command-line tool(s)
* Data portal access for management, search, and transfer

While the SDMS will be developed at, and for, ORNL (initially OLCF and CADES), the system design will not preclude cross-facility deployments. Remote users will be able to utilize the SDMS for data access and dissemination regardless of whether their home facility is integrated into the SDMS network.

# Architecture

The system architecture is shown in the diagram below. The system consists of a set of services and tools that utilize Globus GridFTP services and Grid Security Infrastructure (GSI). A central data store is provided and is managed by a “Central Service” with custom authentication and authorization modules installed in the associated GridFTP server. The Central Service permits users to access their data as well as specify fine-grained access control rules for their data.

**System Architecture Diagram**

Data transfers are initiated using SDMS command line utilities from within a supported local environment (Globus Toolkit and GSI must be provided and managed by the local facility). Because the SDMS utilizes GSI, users must ensure their proxy certificates are enabled within the environments that they intend to use. (GSI certificate policy is under the control of local facility administration.) The SDMS CLI tools communicate with a facility-local SDMS service that, in turn, coordinates activities with the SDMS Central Service. These tools and services implement a management layer on top of GridFTP that simplifies and enhances data access (caching, concurrency control, aliases, version control, tagging, and metadata ingest/search). A Python client library is provided for direct integration with client applications.

A web service and front-end (Data Portal) is provided to allow users to browse and search for data, manage access control, and perform configuration activities. SDMS admins may also use the web service to manage accounts and general policies.

# Scope

The following scope discussion references the system architecture diagram, above. Green items are targeted for tier 1, yellow for tier 2, and orange for tier 3. Gray items are preexisting require no development effort.

**Tier 1 – Basic Data Access and Transfer Support**

1. Configuring central data store and corresponding GridFTP server.
2. Configure local GridFTP server(s) (if needed)
3. Develop data model and deploy central database
4. Develop GridFTP GSI Authn module
5. Develop GridFTP GSI Authz module
6. Development of a central service to manage data (i.e. assign IDs and aliases, control ACLs, manage authentication/authorization)
7. Development of a service to integrate and synchronize local user-level access with central system – primarily for Globus access tokens and potential for security or network constraints would prevent user-level processes from access a centralized service.
8. Develop basic Python client library
9. Develop command line tool(s) for basic data access (create, read, update, delete)

Tier 1 will achieve the data management aspects of Crossbow (only generalized) but without data indexing or a portal.

**Tier 2 – Advanced HPC Features**

1. Implement local data caching and synchronization (local service)
2. Implement version control (central service)
3. Implement subscriber notification (central service)
4. Enhance Python client library with advanced features
5. Enhance command line tool(s) for advanced features (aliases, acls, tagging, etc)

**Tier 3 – Metadata Indexing and Data Portal**

1. Implement metadata ingest API and tools (not metadata extraction)
2. Configure/deploy metadata index (assuming built-in indexing of central DB is insufficient)
3. Integrate metadata search into CLI tools and Python library
4. Either A) select, modify, and integrate an existing data portal (such as CKAN), or 2) develop a data portal from scratch