OOI Data Exchange (DX)  
Prototype Architecture and Technologies

Matthew Armento, Alex Clemenza, Claudia Farias, Paul Hubbard, Michael Pleininger, Dorn Rayner  
AXIS): Code in collaboration with NOAA

The goal of the Data Exchange is to make oceanographic science easier by accelerating the transfer of data, providing cloud-based server-side computation and the propagation of standards for data formats and conventions. Whenever possible, we reuse existing work, adhere rigorously to standards and produce open-source code for the community. The code should scale from a small system that can run on a laptop to a multi-continent archipelago of caches, servers, exchange points and networks. We expect to be moving large amounts of data, and investigated different encoding and transport methods. DAP, with its underlying use of XDR, is quite bandwidth efficient and widely used for remote transport across large datasets (see Signell et al. 2008). Layered on top of DAP are standards and programs that allow transformation, aggregation, computation, and visualization (i.e., Thredds, Ferret, NetCDF, COARDS, GridFields, Matlab, Eddap, pydap and many others).

Overview

The prototype implementation of the OOI Data Exchange (DX) system, depicted in Figure 1, targets the common use case where a researcher using a desktop analysis package, such as MATLAB, needs access to several OpenDAP datasets on remote servers. We assume that the DAP servers and MATLAB code are preexisting and immutable, meaning that neither can be changed. The grey cloud shows the boundaries of OOI and represents the DX ‘cloud.’

The Controller is the most complex component, handling inter-component communication, state management and most of the algorithmic complexity. When it gets a DAP request, it queries the Attribute Store to see if the dataset has already been cached. If so, it forwards the DAP request to the cache and sets the reply-to address to a temporary topic it created for the proxy. If the dataset isn’t found, the Controller must tell the Persister to begin a new dataset, and instruct a Fetcher to fetch it from the original server.

The Persister assembles and serializes the dataset into a local netCDF file, which is then auto-indexed and served by the Cache. Once completely cached, the Controller updates the cache list in the Attribute Store. The Controller creates and forwards new messaging addressed to various components, e.g. ‘Proxy, please listen for your data on topic X.’ Fetcher, please transmit the DAP data to routing key X.’

Controller handles cache misses. It does not initiate transfers and is purely a listener. Persisters are paired with a Cache process, over any interconnect that allows the sharing of disk. The serialization strategy used by the Persister is to create pydap objects from the data servers via pydap’s netCDF persister. This lets us reuse known-good code and formats (see Figure 2).

Management Interface

The management interface is a Web front-end (Figure 3) that communicates with various components, primarily the Controller. It allows the administrator to set policies on datasets, e.g. ‘Never cache Doppler radar datasets from LEAD’, examine the contents, add or remove datasets and other functions.

Attribute Store

The attribute store is an AMQP-native key-value store, currently written to the Redis back-end, that is used to store object persistent state. The DX uses it to keep track of datasets and their status. The DAP dataset being used by the user may in fact be created on the fly by a Ferret or Gridfields program, or other server-side engine. The DX supports this transparency, as all data is still specified via (admittedly complex) DAP URLs and returned as DAP data.

Cache

The Cache will be lightly modified open-source software, where we add the ability to serve new datasets without restarting or reindexing directories. We also need the ability to delete less-used datasets if disk fills up. Pydap, Thredds or others are possible here.

Supporting Infrastructure

The Messaging System is part of the OOI-CI CCI subsystem and is responsible for abstracting the communication infrastructure, decoupling and integrating the DX (and other CI) components. It implements a federation of brokers that exchange messages asychronously using the concept of queues (see the PS poster for details).

Cloud Computing

In order to realize a scalable deployment of capability containers across the OOI network, the CI applies virtualization of computation and storage, which can be provided on demand over the Internet.

Collaboration with NOAA IOOS

The NOAA Integrated Ocean Observing System (IOOS) Data Integration Framework (DIF) has established an experimental deployment of standardized information access services, including OPeNDAP, OGC Sensor Observation Service (SOS), and OGC Web Map Service (WMS). In addition to the capabilities provided by the 2008 version (i.e., web-based access to data through NOAA’s EDDAP, efficient science metadata caching, scalable deployment using Amazon’s EC2), this prototype provides a server-side data processing capability for use by an initial set of ocean modeling communities (NERACOOS, MARCOOS, SCCOOS) to efficiently exchange large model datasets in whole or in part, while preserving the original content and structure of the dataset. An example of using the prototype to access remote data with Google Earth is depicted in Figure 4.

Figure 1. Prototype Deployment Architecture

Figure 2. Pydap + WMS

Figure 3. Management Interface

Figure 4. Results view using Google Earth