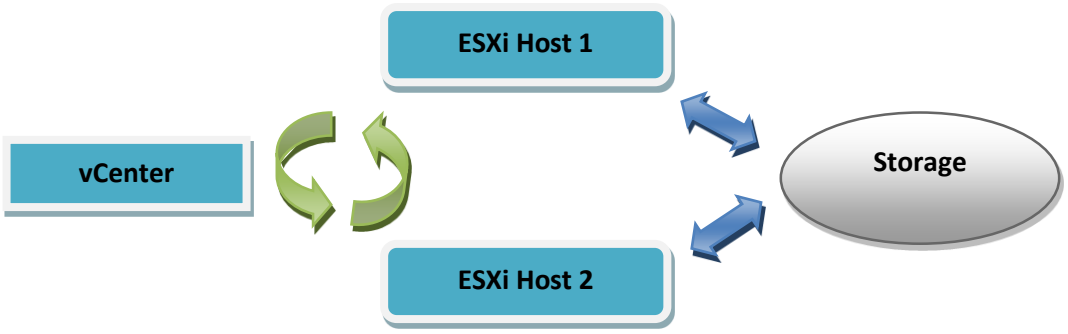
GLOS Technology Stack

1. Server Configuration

a. Hardware

GLOS currently has two ESXi virtual hosts. The first is a DELL PowerEdge R720 with two 8-Core (HT) Intel Xeon CPUs and 96 GB RAM. This server is fully operational and serves as the backbone for GLOS DMAC operations. The other virtual host is a DELL PowerEdge R710 with two 4-Core Intel Xeon CPUs and 32 GB RAM, used for development and load balancing. The diagram below shows a generalized view of the planned server structure at GLOS. The two virtual hosts on the network running VMware VSphere are managed and maintained through an administrative node with vCenter, which is installed on a HP PROLIANT DL360 with a 4-Core Intel Xeon CPU and 12 GB RAM. vCenter offers a centralized application that is able to access and control multiple ESXi hosts with one interface across network. The virtual hosts access the GLOS datasets on the main GLOS storage appliance (DELL PowerVault MD1200 with 12 1TB 7200K RPM SAS drives in a RAID 5 disk array) using the NFS protocol through a directly-attached DELL PowerEdge 2950 with two 4-Core Intel Xeon CPUs and 16 GB RAM. This NFS approach has been identified as a likely bottleneck for large-scale data access and GLOS DMAC team is working on a solution to boost the throughput.

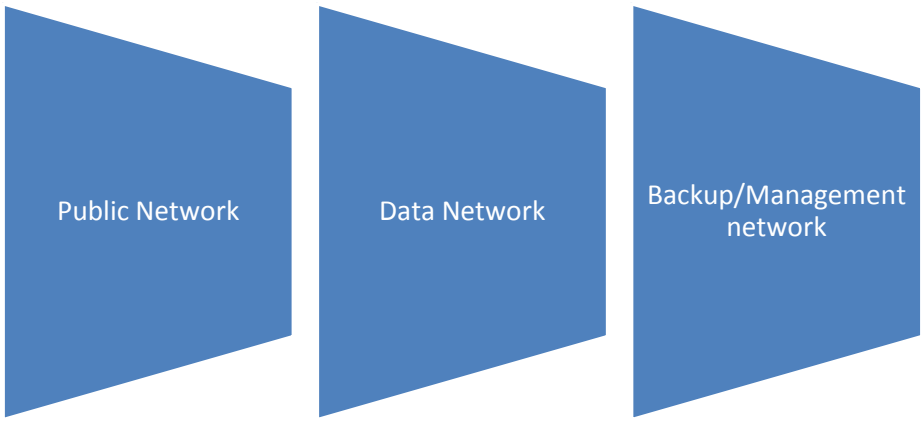


A table of the GLOS server/appliance specifications is provided below. All servers, including virtual guests, are running against RedHat Enterprise or CentOS, Linux operating system (5.X, 6.X and 7.X), except for the HP PROLIANT DL360, which is configured with Windows Server 2008 R2.

Model	CPU	RAM	Disk	RAID Level	OS
DELL PowerdEdge R720	2 X Intel Xeon CPU E5-26650 @ 2.40GHz	96GB	4 X 600GB SAS 15k rpm	5	ESXi
DELL PowerEdge R710	2 X Intel Xeon CPU E5520 @ 2.27GHz	32GB	4 X 300GB SAS 15k rpm	5	ESXi
DELL PowerEdge 2950	2 X Intel Xeon CPU E5405 @ 2.00GHz	16GB	2 X 150GB SAS 15k rpm	1	RedHat Enterprise Linux 5.9
HP PROLIANT DL360	1 X Intel Xeon CPU E5606 @ 2.13GHz	12GB			Windows Server 2008 R2
DELL PowerVault MD 1200	n/a	n/a	12 X 1TB SAS 7.2k rpm	5	n/a

b. Network

The GLOS network was designed and built for tolerance on high I/O throughput in order to support massive data transfer. A three-tier approach was adopted to address this requirement. As depicted below, the approach calls for a public sub-network for data access from Internet, with data and the management/backup sub-networks visible only within the rack space.



With a three-tier network, the traffic can be split onto dedicated switches, thereby preventing the congestion and interference of a single combined network. The two local networks are linked with a Cisco 36-port Gigabit managed switch, and the public network has a 3Com 48-port 100Kb managed switch, which will be upgraded by 2016 to gigabit.

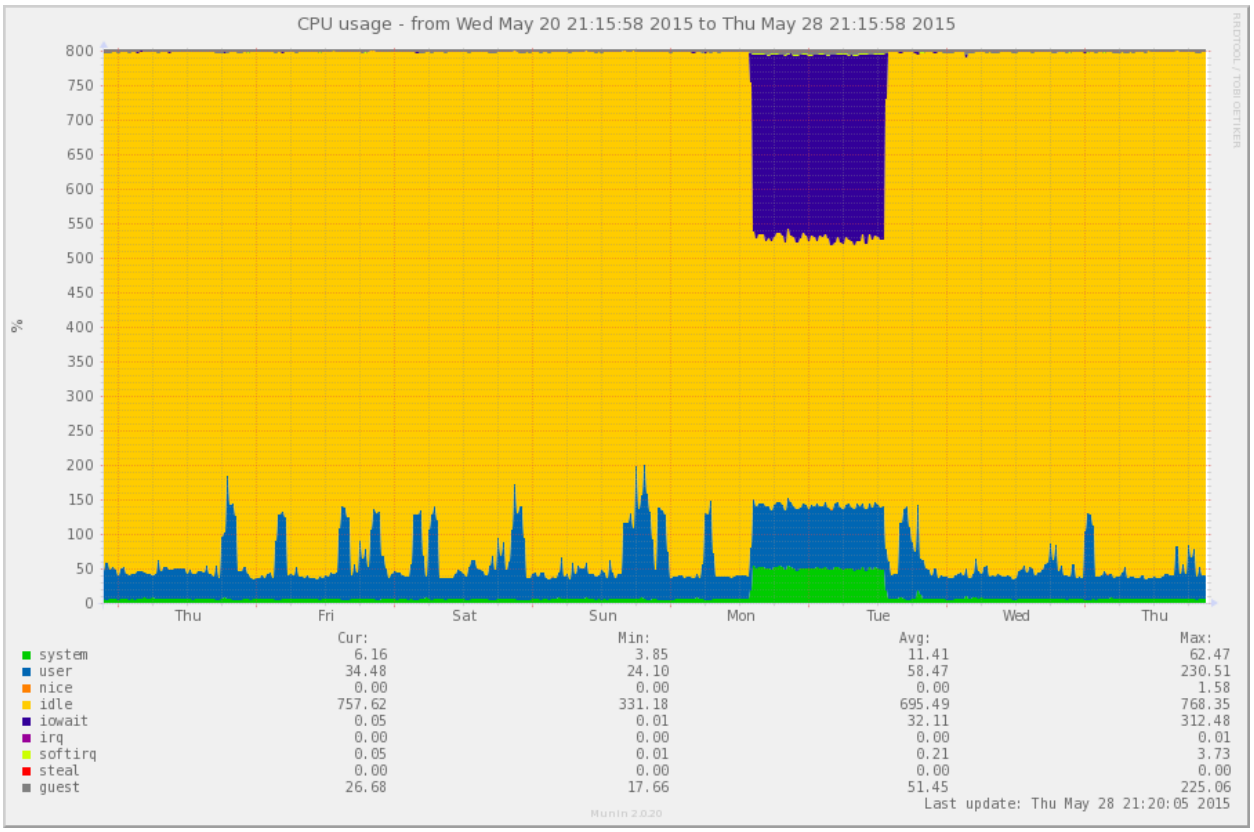
Due to the limited numbers of public IP addresses available to GLOS servers, one virtual machine was configured to act as a gateway. Other virtual machines which typically conduct backend processing work take advantage of this software gateway for their occasional Internet access.

c. Backup/Recovery

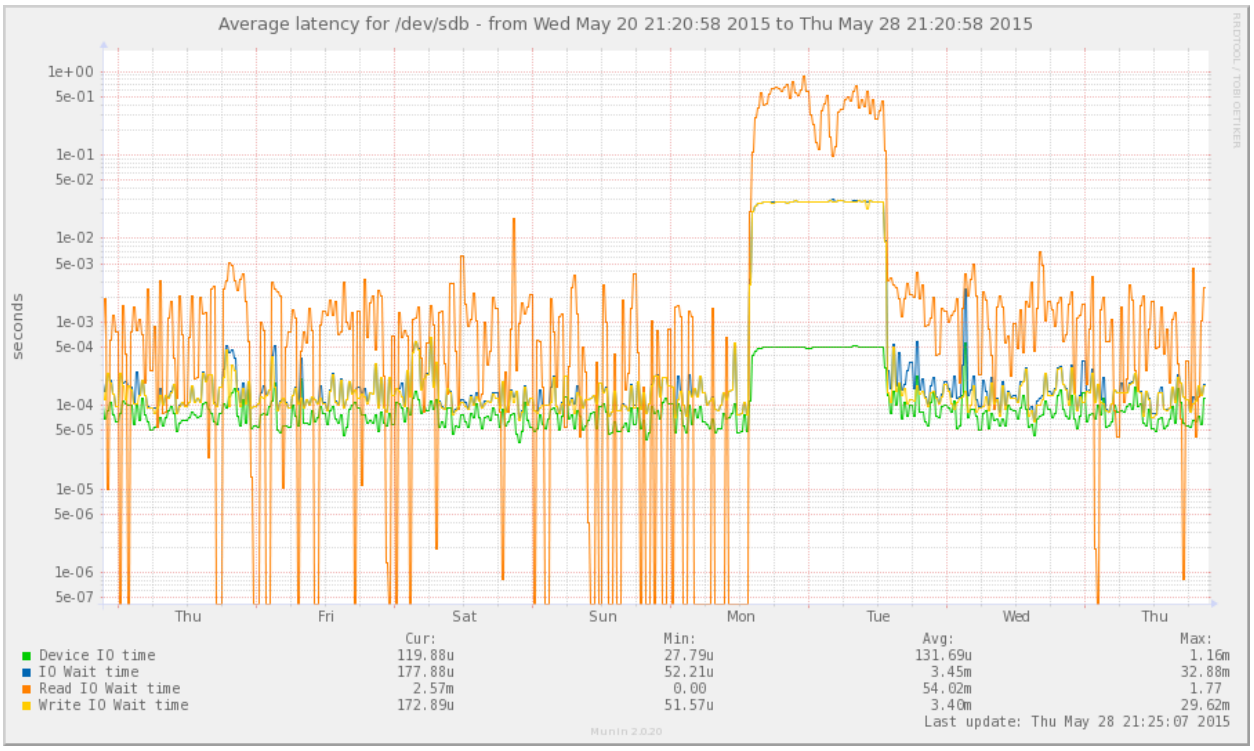
Backups are made of virtual machines through a dedicated network which will not be interrupted by other data access I/O operations. Snapshots are currently taken manually with Veeam; by November 2015, GLOS will deploy Veeam Enterprise and additional hardware to support automated daily snapshots as well as failover.

d. Monitoring

GLOS relies on metrics collected through monitoring software to monitor and track the performance of the hardware, software and network. The collected information is used for planning or other purposes that depend on reliable, continuous data harvesting. Munin is the tool GLOS currently uses to view and chart server metrics. It gives administrators near real-time charting on various parameters that are vital for the operation of hardware and software. For example, the chart below, shows a high I/O wait on CPU usage. This indicates a very heavy I/O operation was conducted during that period and we would expect to see a similar pattern on the chart of disk IO or network traffic.



The next plot shows disk I/O per second and confirms that heavy I/O operations were performed on the local disk array. More specifically, there was a huge delay on I/O write; by backtracking down the line, the original cause can be discovered.



With this capacity, the administrator at GLOS can not only monitor the performance of servers visually, but conduct troubleshooting more efficiently as well. However, Munin does have some limitations, so Zabbix will be deployed by December 2015 to complement Munin. Zabbix is an open-source enterprise monitoring solution that offers a dashboard-like GUI to conduct not only monitoring but near real-time notification/alert through triggers as well.

e. Operating System/Software

GLOS relies heavily on open source software. Almost all operating systems are Linux: GLOS has RHEL 5.9, CentOS 6.6 and CentOS 7.1 deployed on either virtual or physical machines. These operating systems offer good performance, community-supported documentation and extensive knowledge bases. More importantly, GLOS has virtually unlimited access to thousands of free open-source software packages built for the Linux ecosystem. Among these are Apache and nginx web servers, Tomcat application server, and MySQL and PostgreSQL databases, which underlie the mainstream application packages on GLOS servers. GLOS also has deployed the Unicorn python http server, Ruby on Rails, and MongoDB as for certain applications that were developed to meet specific demands.

GLOS also recognizes the benefits of using Microsoft Windows servers onboard for certain tasks which could be tackled more efficiently using a GUI environment, such as ESXi administration through

vClient/vCenter. GLOS has a diverse and robust OS/software environment that is capable to meet the challenges of data management. GLOS OS and key software are tabulated per host below:

Hostname	**OS**	**Virtual machine**	**Role**	**Application Environment/Server/Database**
process1.glos.us	CentOS 7.1	Yes	Hosting backend scripts for data harvesting, data consolidation, data delivery	■ C ■ Lua ■ Python ■ Java ■ Scala
gateway.glos.us	CentOS 6.6	Yes	Software gateway for backend machines accessing Internet	
db1.glos.us	CentOS 6.6	Yes	PostgreSQL Database Server	■ PostgreSQL ■ PostGIS
db2.glos.us	CentOS 6.6	Yes	MySQL Database Server	■ MySQL/MariaDB
gn25.glos.us	CentOS 6.6	Yes	JEE container	■ Tomcat 7 ■ IOOS 52n SOS ■ GeoNetwork
thredds.glos.us	CentOS 6.6	Yes	JEE container	■ Tomcat 7 ■ THREDDS
wms.glos.us	CentOS 6.6	Yes	Multiple roles: sci-wms, baseX database	■ Python virtualenv ■ Python Gunicorn HTTP server ■ BaseX Nosql Database
web1.glos.us	CentOS 6.6	Yes	Web Application Server	■ Python virtualenv ■ Python Gunicorn HTTP server
web2.glos.us	CentOS 6.6	Yes	Web Application Server, Drupal	■ LAMP
glos.us	RHEL 5.9	No	Multiple roles: Web Application Server FTP Server NFS Server Database Server	■ LAMP ■ GLOS Portal ■ Ruby on Rail ■ vsftpd ■ nfsd ■ PostgreSQL ■ MySQL
back.glos.us	Window Server 2008 R2	No	Management Console	■ vSphere vClient ■ vShpere vCenter

GLOS is currently working on the migration of applications and databases from the physical host "glos.us" onto virtual hosts. This task will be completed by the end of 2015.

f. Data File Structure on File System

GLOS stores and archives observation, remote sensing and numeric model data in any flat file format on a dedicated disk array that is directly attached with a server to offer data read (NFS) and write (scp) services for various applications. There are currently 6 TB of storage on the file system. A self-explanatory file structure is maintained for easier management and faster data retrieval, with top-level directories on the file system are named after the datasets as listed below:

Directory Name	Data Collection/Time Span	Format	Size
Bathymetry	Great Lakes bathymetry	NetCDF	Overall 1.1GB
GLCFS	Great Lakes Coastal Forecasting System, including nowcast, forecasting for the current year and archival from 2006 to current.	NetCDF	Overall 4.3TB Nowcast current year: 28G Forecast current year: 394G Nowcast Archival: 547G Forecast Archival: 3.4T
glider	Data collected through glider operation around Great Lakes	NetCDF	844MB
GLOP	Great Lakes Optical Properties in 2014	NetCDF	3.8MB
GLSEA	Great Lakes Surface Environmental Analysis data from 2008 to current	tif	5.0GB
HABS	Water quality in-situ data on Lake Erie	NetCDF	7.7MB
HECWFS	Huron Erie Corridor Way Forecasting System from 2010 to current	NetCDF/shp	Overall 814GB Nowcast: 185G Forecast: 536G Archive: 50G Shp file: 44G
MTRI-CDOM	Colored dissolved organic matter data from 2013 to current	NetCDF	453MB
MTRI-CHL	Lake Surface Chlorophyll data from 2013 to current	NetCDF	470MB
MTRI-DOC	Dissolved Organic Carbon data from 2013 to current	NetCDF	425MB
MTRI-LST	Lake Surface Temperature from 2013 to current	NetCDF	35GB

MTRI-NC	Natural Color Imagery from 2013 to current	NetCDF	776MB
MTRI-Ranger3	Ranger 3 ferry observation data in 2013	NetCDF	17MB
MTRI-SM	Suspended Minerals in 2013	NetCDF	455MB
SLRFVCOM	St. Lawrence River FVCOM Forecasting System from 2012 to current	NetCDF/shp	Overall 810GB Nowcast: 69G Forecast: 738G Shp file: 3.2G
USACE-WaterLevel	USACE Water Level report from 2014 to current	NetCDF	1.7MB

At the sub-directory level, data are organized either by the type of data or the geographic extent. For example, the Great Lakes Forecasting System (GLCFS) has a sub-directory structure:

- Archive
- Forecast
- Forecast-archive
- Nowcast

GLCFS data are generated on daily basis, new data are pushed in to Nowcast and Forecast folder for immediate access. By the end of the calendar year, these data then will be moved to an Archive folder by year for nowcast data and to a forecast-archive folder. Since the sub-directory structure on the second level can't distinguish the geographic extent of data without checking the metadata or coordinates inside the files, a naming convention/coding schema on the individual file level is maintained. For instance, in the name "e201513000.out3.nc", the first letter 'e', which indicates Lake Erie, and the following digits represent the initial date for the model time period in the file. Date/time is represented in a format YYYYDOY (year and the day of the year). 'out3' means this is the output for the 3 dimensional model. The complete naming conventions are as follows:

'e': Lake Erie

'h': Lake Huron

'm': Lake Michigan

'o': Lake Ontario

's': Lake Superior

'in1': forcing data (input)

'out1': 2 dimensional model output

'out3': 3 dimensional model output

Other numeric models, such as HECWFS and SLRFVCOM, have the same sub-directory structure as GLCFS. However, since they are running at a localized level, all of the naming conventions are applied except for the leading letter, which is not necessary for a model that is running on a specific geographic extent. The top level directory is sufficient to indicate the spatial extent.

There are also datasets organized by geographic extent directly at the sub-directory level. For example, the USACE water level data has a secondary folder structure:

- GreatLakesWaterLevels
- LakeErieWaterLevels
- LakeOntarioWaterLevels
- LakesMichiganandHuronWaterLevels
- LakeSt.ClairWaterLevels
- LakeSuperiorWaterLevels

The directory structure on the GLOS disk array is determined not only by the nature of the data, such as spatial extent, update cycle, data format, but application demand as well. Some datasets, such as water quality data (HABS) were created to serve specific application requirements.

g. Backend Scripts

There are three types of scripts that exist inside GLOS DMAC:

1. Data/metadata harvest, consolidation, and persistence;
2. Status and performance monitoring on scripts and applications;
3. Data/metadata processing and preparation for applications or data archival;

Data/metadata harvest and processing is one of the fundamental tasks of GLOS DMAC. The functions provided by these scripts drive the entire GLOS' data infrastructure. In an ideal world, data are all written in standard formats and follow standard protocols. Therefore, a single suite of scripts should be able to handle all data related tasks. Unfortunately, this is not the case. Most data that GLOS collects come from divers scientific/research fields, which makes it yet more difficult to enforce standards and protocols due to the uncertainty and complexity of disparate, complex research topics. In this context, scripts might have to be written for every individual dataset, which is obviously not desirable for any data management task. In order to avoid reinventing the wheel again and again during programming, the common features for data harvest/process tasks from these scripts have been extracted and compiled into separate libraries for further maintenance and development in a more coordinated approach.

Python and bash are the primary scripts for the GLOS backend script because they are flexible and versatile. C and Java are also on the stack of programming languages for handling more complicated tasks that require performance over flexibility. If a tool is developed primarily in C or Java, it usually has Python, bash or a Lua wrapper on top as the glue to bind the C or Java components together.

Most GLOS scripts are on a schedule using Linux cron jobs. A migration effort is currently under way to centralize all scripts onto one or two virtual machines equipped with all necessary environments and dedicated for running backend scripts. Processed data are pushed by these scripts to either GLOS storage or to the databases that serve various applications that need data access.

Key GLOS scripts are listed in the table below:

Script Name	**Main Programming Language**	**Purpose**
metadown	Python	Metadata harvesting and compilation from GLOS THREDDS
sensor-web-harvester	Scala	Populate GLOS IOOS 52n SOS server with GLOS buoy data
swh-metadata-iso.sh	Bash/Scala/Python	Metadata harvesting and compilation from GLOS IOOS 52n SOS
glos_obs	Java	Populate GLOS observation database with buoy data from NOAAPORT and GLOS FTP
water_levels	Python	Parsing USACE water level report from PDF to NetCDF
obs2nc	C/Lua	Convert observation data to NetCDF format file that is NCEI compliant
obs2habs	C/Lua	Convert observation data to NetCDF format to serve HABs portal
glcfs_thredds	Python	Harvesting GLCFS model output
hecwfs.py	Python	Harvesting HECWFS model output
slrfvm.py	Python	Harvesting SLR FVCOM model output
Obscan	PHP	Buoy data update check
Status	PHP	Data harvesting job status check
glsea.py	Python	Harvesting GLSEA output
Slrfvm	C	Convert SLRFVCOM model output from NetCDF to shp file
mtri-img.sh	Bash	Populate GLOS storage with remote sensing data residing on GLOS FTP

Data flows to GLOS using either push or pull mode. Data from GLOS sponsored projects, such as observation data and remote sensing data, are using push mode. A password protected FTP site was set up to accept the data from GLOS partners. For example, GLOS currently has 36 buoys that have data actively pushed onto FTP. In order to coordinate data collection effort, an XML schema was defined for

buoy data contributors. This schema is widely adopted in the Great Lakes Region and supported through the software from buoy manufacturers.

The schema is defined as follows:

1. **XML Schema**

```
<station>Station Name</station>
 <date>MM/DD/YYYY HH:MM:SS</date> <!—Timestamp must be UTC -- >
 <met>
      <!—envelop for the observation data -- >
 </met>
```

2. **XML Tags**

Note: This is a current list as of this version of the Enrollment Guide. Please check the GLOS website to obtain an update or contact GLOS DMAC staff (Guan Wang, gwang@glc.org) to establish new tags that accommodate your data.

<wdir1>	Wind Direction
<wspd1>	Wind Speed
<gust1>	Wind Gust
<atmp1>	Air Temperature
<wtmp1>	Water Temperature
<rh1>	Relative Humidity
<dewpt1>	Dew Point
<baro1>	Air pressure
<wvhgt>	Significant Wave Height
<dompd>	Wave Period
<mwdir>	Wave Direction
<dp###>	Depth in Water for Nodes on a Thermal String (e.g., dp001, dp002, etc.)

<tp###> Water Temperature for Nodes on a Thermal String (last three digits correspond with the matching dp### value)

3. Enclosure Structures

Single timestamp format:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<message>          <!—envelop for  obs data for a timestamp  -
- >

   <!—XML Schema for Data -->

   <station>Station Name</station>

    <date>MM/DD/YYYY HH:MM:SS</date> <!—Timestamp must be UTC
   -- >

    <met>

        <!—envelop for the real data -- >

    </met>

</message>
```

Multiple timestamp format:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<messages>    <!—envelop for obs data for multiple timestamp  -
- >

   <message>

      <!—XML Schema for Data -->

      <station>Station Name</station>

      <date>MM/DD/YYYY HH:MM:SS</date> <!—Timestamp must be UTC
      -- >

      <met>

         <!—envelope for the real data -- >

         </met>
```

```
    </message>
    <message>
      <!—XML Schema for Data -->
      <station>Station Name</station>
      <date>MM/DD/YYYY HH:MM:SS</date> <!—Timestamp must be UTC
      -- >
      <met>
        <!—envelope for the real data -- >
      </met>
    </message>
</messages>
```

Below are samples of GLOS Observation Data in XML format:

1. Thermistor string-equipped buoy, single timestamp

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <message>
      <station>45025</station>
      <date>06/22/2011 22:40:00</date>
      <met>
         <wdir1>64.92</wdir1>
         <wspd1>10.25</wspd1>
         <gust1>13.2</gust1>
         <atmp1>7.979001</atmp1>
         <wtmp1>8.8</wtmp1>
         <rh1>97.8</rh1>
         <dewpt1>7.650001</dewpt1>
```

```
            <baro1>1002</baro1>
            <wvhgt>1.487</wvhgt>
            <dompd>7.014001</dompd>
            <mwdir>22.59</mwdir>
            <dp001>4</dp001>
            <tp001>8.729999</tp001>
            <dp002>6</dp002>
            <tp002>8.74</tp002>
            <dp003>9</dp003>
            <tp003>9.66</tp003>
            <dp004>11</dp004>
            <tp004>8.61</tp004>
            <dp005>13</dp005>
            <tp005>8.45</tp005>
            <dp006>15</dp006>
            <tp006>7.857</tp006>
            <dp007>17</dp007>
            <tp007>8.61</tp007>
            <dp008>19</dp008>
            <tp008>8.82</tp008>
         </met>
      </message>
```

2. Thermistor string-equipped buoy, multiple timestamps

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<messages>
   <message>
```

<station>45025</station>

<date>06/22/2011 22:40:00</date>

<met>

<wdir1>64.92</wdir1>

<wspd1>10.25</wspd1>

<gust1>13.2</gust1>

<atmp1>7.979001</atmp1>

<wtmp1>8.8</wtmp1>

<rh1>97.8</rh1>

<dewpt1>7.650001</dewpt1>

<baro1>1002</baro1>

<wvhgt>1.487</wvhgt>

<dompd>7.014001</dompd>

<mwdir>22.59</mwdir>

<dp001>4</dp001>

<tp001>8.729999</tp001>

<dp002>6</dp002>

<tp002>8.74</tp002>

<dp003>9</dp003>

<tp003>9.66</tp003>

<dp004>11</dp004>

<tp004>8.61</tp004>

<dp005>13</dp005>

<tp005>8.45</tp005>

<dp006>15</dp006>

<tp006>7.857</tp006>

<dp007>17</dp007>

<tp007>8.61</tp007>

<dp008>19</dp008>

<tp008>8.82</tp008>

</met>

</message>

<message>

<station>45025</station>

<date>06/22/2011 22:50:00</date>

<met>

<wdir1>64.92</wdir1>

<wspd1>10.25</wspd1>

<gust1>13.2</gust1>

<atmp1>7.979001</atmp1>

<wtmp1>8.8</wtmp1>

<rh1>97.8</rh1>

<dewpt1>7.650001</dewpt1>

<baro1>1002</baro1>

<wvhgt>1.487</wvhgt>

<dompd>7.014001</dompd>

<mwdir>22.59</mwdir>

<dp001>4</dp001>

<tp001>8.729999</tp001>

<dp002>6</dp002>

<tp002>8.74</tp002>

<dp003>9</dp003>

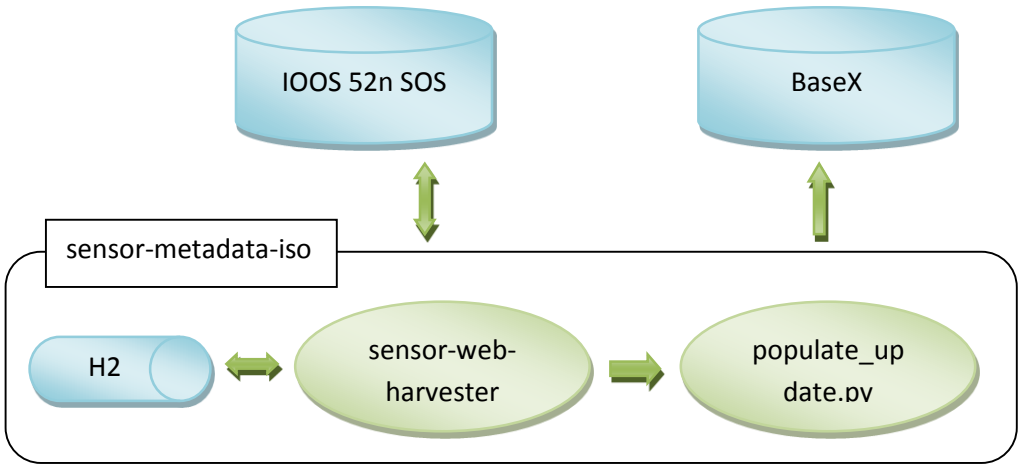
<tp003>9.66</tp003>

<dp004>11</dp004>

```
            <tp004>8.61</tp004>
            <dp005>13</dp005>
            <tp005>8.45</tp005>
            <dp006>15</dp006>
            <tp006>7.857</tp006>
            <dp007>17</dp007>
            <tp007>8.61</tp007>
            <dp008>19</dp008>
            <tp008>8.82</tp008>
        </met>
    </message>
</messages>
```

Once the observation data arrive at GLOS via FTP, they are picked up by the Java package glos_obs. This acts as an adaptor between data in XML format and the GLOS observation database. The package is comprised of two major components: an XML data parser and a database injector. The parser scans the FTP folders, analyzes the structure of theXML files, and extracts validated records into java pojo. Then the java pojo collection is passed to database injector, which prepares and inserts observation data into the database. Before the XML file is removed from FTP, a copy of the file is transferred to a separate folder where the sensor-web-harvester package scans the data and populates GLOS IOOS 52n SOS.

The metadata record is as important as the data itself in order to make data discoverable. GLOS' smetadata search is powered by BaseX, which is populated using a Metadown package and swh-metadata-iso package. Metadown harvests the metadata records on GLOS THREDDS through the ncISO service, and Swh-metadata-iso exports the metadata record from GLOS IOOS 52n SOS. The generated xml metadata records then populate the BaseX database. The diagram below shows this procedure for updating a metadata record through swh-metadata-iso.



NetCDF is a very popular format that is widely accepted by the IOOS community. Manipulating data in NetCDF format is a common task for many data related operations. However, the official NetCDF libraries only support C and Java. There are community editions of Python port NetCDF library, but GLOS has identified compatibility issues that prevented us using it for certain scenarios. The NetCDF C library, on the other hand, is a native library that is able to work through every aspect of NetCDF.

Compared as a system level programming language with Python, C has difficulty handling string/text. Therefore, an auxiliary scripting tool, Lua, which can work with C seamlessly, is used to address the gap. For instance, GLOS has a package called obs2nc which was developed for converting observation data from the GLOS observation database to NCEI-compliant NetCDF for long term archival purpose. The entire NetCDF creation routine was done using C, but the configuration and metadata population for buoys was developed in Lua. The processed information handled by Lua is used as feed to drive the routines written in C. By using Lua on top of C, a balance between performance and flexibility can be found.

Below is an example of the definition of a GLOS buoy in Lua. It's human-friendly and can be maintained and populated by staff who may only a limited understanding of programming:

```
Platform={
  id="45167",
  title="NOAA_RSC_A",
  lon=-80.14,
  lat=42.19,
  summary="Regional Science Consortium buoy",
  keywords="GLOS,Regional Science Consortium,Lake Erie",
  sensors={
    sea_surface_water_temperature={
        standard_name="sea_water_temperature",
        long_name="sea water temperature at surface",
        featureType="timeSeries",
        units="degree_Celsius",
        source="platform/45167/45167_sea_surface_water_temp",
        depth=0.0,
```

```
        keywords="EARTH SCIENCE > OCEANS > OCEAN TEMPERATURE > SEA SURFACE
TEMPERATURE",
        keywords_vocabulary="GCMD Earth Science Keywords. Version 5.3.3",
        validator=function(val)
                if val>100 or val<-10 then
                   return false
                else
                   return true
                end
             end
      },
      sea_water_temperature={
        standard_name="sea_water_temperature",
        long_name="Thermistor water temperature",
        featureType="timeSeriesProfile",
        units="degree_Celsius",
        source="platform/45167/45167_sea_water_temp",
        keywords="EARTH SCIENCE > OCEANS > OCEAN TEMPERATURE > WATER TEMPERATURE",
        keywords_vocabulary="GCMD Earth Science Keywords. Version 5.3.3",
        comment="",
        validator=function(val)
                if val>100 or val<-10 then
                   return false
                else
                   return true
                end
```

```
            end
        },
        …. ….
    }
}
```

From this snippet of Lua code, it can be observed that its table structure can define all necessary components for describing a NCEI-compliant buoy. The QA/QC procedure, “validator”, can be included in the table and then invoked in a C routine.

GLOS DMAC operations heavily rely on automated script tools for data management. By taking advantage of various programming languages and existing tools, GLOS keeps a robust and flexible script suite that is capable to handle all kinds of data driven tasks.