CO_2DAS Data Ingestion System Design Version 1.0

Charles J Antonelli

1 April 2011

Introduction

The CO_2DAS Data Ingestion System (DIS) is responsible for discovering and remembering sources of data, and ensuring that new data appearing at a source are staged to local disk for storage and subsequent processing[1]. This document describes the design of the Data Ingestion System.

1.0 Source Control Component

Allows users to add, delete, and modify sources of data, access parameters (e.g. polling frequency), and parameters for the assimilation, surveillance, and visualization framework stages. These metadata are maintained in the Master List.

1.1 Master List

The Master List is maintained in a MySQL database. Each source description is stored as a row in a single table. We expect fewer than a thousand sources; using a single table will not impact scalability. The schema is:

```
CREATE TABLE MASTER(
Source_Id         INT PRIMARY KEY AUTO_INCREMENT,   -- Primary key
Source_Name       VARCHAR(4096) NOT NULL,           -- Name of this source
Source_DocURI     VARCHAR(4096) NULL,               -- URI of source doc, if any
Source_LastUpdate DATETIME NULL,                    -- Time of last known update
Source_NextUpdate DATETIME NULL,                    -- Time of next known update
Source_URI        VARCHAR(4096) NOT NULL,           -- URI of source root
Source_Dir        VARCHAR(4096) NOT NULL,           -- Source dir within root
Source_File       VARCHAR(4096) NOT NULL,           -- File(s) within source dir
Source_Format     ENUM(                             -- Source format:
                       'HDF5',                      --  HDF5
                       'NetCDF',                    --  NetCDF
                       'TEXT') NOT NULL,            --  Plain text
DIS_Dir           VARCHAR(4096) NOT NULL,           -- Cache dir within DIS
DIS_Format_HDF5   ENUM('y','n'),                    -- Store a copy as HDF5 within DIS
DIS_Format_Binary ENUM('y','n'),                    -- Store a copy as binary within DIS
DIS_Format_Text   ENUM('y','n'),                    -- Store a copy as text within DIS
Agent_Type        ENUM('FTP','HTTP','RSS') NOT NULL,-- Agent required
Status            ENUM(                             -- Source status:
                       'I',                         --  Initialized
                       'D',                         --  Downloaded
                       'F',                         --  Formatted
                       'T',                         --  Transformed
                       'R',                         --  Ready (& sched for acq)
                       'B',                         --  Being accessed by agent
```

[1] A.M. Michalak et al, "SI2-SSI: Real-Time Large-Scale Parallel Intelligent CO_2 Data Assimilation System," Section 4.4.

```
                        'C') NOT NULL,              --  Completed
Status_LastOp     DATETIME NOT NULL,                -- Time status last changed
Status_BatchID    VARCHAR(4096),                    -- Scheduled batch job identifier
Callback_Spec     VARCHAR(4096),                    -- Callback specification
Source_PollFreq   INT,                              -- Poll frequency, hours
Source_LastPoll   DATETIME NOT NULL                 -- Time source last polled
) ENGINE = InnoDB;
```

A source may consist of one or more data files. The `Source_URI` specifies a remote server, `Source_Dir` specifies a remote directory on the server, and `Source_File` specifies the source file(s) on the server; of these, only the `Source_File` name may contain wildcards.

1.2 Source Validation

Source validation involves accessing a new or modified source description and validating the remote URI, the path to the remote data directory, and existence of the files themselves. Detailed results of the verification are recorded in a log.

The DIS realizes source validation by marking the source as initialized and scheduling an acquisition agent for immediate execution. We will employ the Torque (or similar) scheduler for these scheduling operations.

2.0 Acquisition Component

Acquisition agents access remote data and download them, using a polling schedule defined for each source, and an appropriate plugin as identified in the Master List.

Each agent is an autonomous entity and runs in a separate process. Several agents may execute concurrently; appropriate locking primitives are used to ensure consistency.

At the scheduled time, an acquisition agent marks the source busy and invokes the proper plugin for the source; when the plugin is finished, the agent accesses the Master List, computes when the source should next be polled, and schedules an acquisition agent to run at that time. It then marks the source as downloaded and schedules a formatting agent for immediate execution.

To guard against inconsistent source content (e.g., simultaneous update while acquiring), the agent will record the file size and modification time before and after acquisition, and repeat the acquisition if they are found to differ. If this discrepancy occurs more than a configured number of times, the agent abandons the acquisition and notes the event in the log

In this manner only one acquisition agent per source runs at any given time, and each such agent schedules its own next invocation. Source validation starts the first agent.

An acquisition agent determines whether new data exist at the server by comparing them to data previously fetched and stored in the DAS. As storing the data for comparison cannot scale, a modification time and size, and possibly a hash, of each data file is stored instead.

The DAS_Dir parameter specifies a caching directory within the DAS, which holds in distinct subdirectories the original downloaded data as well as formatted and transformed versions of the source data.

If the source is not in either the ready or the initialized state when the acquisition agent starts, a data overrun has occurred; this is noted in the log, and the acquisition agent re-schedules itself several hours into the future. Alternatively, the agent could create a new instance of a caching directory and download there; while useful for short-term overloads, it is unlikely that the software pipeline could "catch up" in the steady state.

2.1 Acquire (FTP)

The FTP plugin uses the File Transfer Protocol to access the specified FTP server URI, establishes a connection, changes to the specified directory, and fetches the specified file(s).

2.2 Acquire (HTTP)

The HTTP plugin uses the HTTP protocol via the `wget` tool to access the specified HTTP server URI, establishes a connection, changes to the specified directory, and fetches the specified file(s).

2.3 Acquire (RSS)

TBD

3.0 Format Component

This component, if required, alters the format of data downloaded to the DIS to the format(s) specified. A separate copy of the data is retained in the DIS for each specified storage format.

When invoked, a formatting agent marks the downloaded data busy and invokes each specified formatting plugin for the data; when all plugins have finished, the agent marks the source as formatted and schedules a transform agent for each specified format for immediate execution. If the downloaded data format matches a specified storage format, that formatting plugin is obviated.

4.0 Transform Component

This component, if required, transforms data downloaded to the DIS to 1x1 degree granularity.

When invoked, a transform agent marks the formatted data busy and invokes the proper transform plugin for the data; when the plugin is finished, the agent marks the data as ready and schedules a callback agent for immediate execution. If the formatted data is already at 1x1 degree granularity, a null plugin is invoked, which causes the formatted data to be interpreted as the transformed data.

5.0 Callback Component

Starts the assimilation, surveillance, and visualization components when new data is available for them.

5.1 Callback Initiator

This component takes the form of a UI application that allows the user to specify, for a given Data Assimilation System application, when sufficient new data have accumulated in the DIS to allow a particular DAS application to run. For example, an application can specify that it should run whenever data for a particular date range have been ingested.

The callback specification language is based on that of crontab(5), extended to support sets of days. The callback initiator validates the specified callback parameters and generates a callback specification, which is stored in the Master List for the source.

The MODIS protocol is used to resolve spans of time that do not fit precisely within an enclosing span. For example, eight-day spans within a year will possibly have a short final span, as it will not cross into the next year; the same span within a month may continue into the next month to complete the span.

The callback specification can include multiple DAS applications and callback parameters, although the initial implementation will be restricted to one.

Examples

Callback specification:

```
# Y     M     D     WY    DY    WM     DW     APP     ARGS
2004    *     *     *     1:8   *      *      pctm    -v

2004 * * * * * *          all data 2004, one dataset
2004 2 * * *              feb 2004, one ds
2004 1 1 * * * *          1/1/2004 only, one ds
2004 2 1-8 * * *          1-8 feb 2004, one ds
2004 2 1/8 * * * *        1,9,17,25 feb 2004, one ds
2004 2 1:8 * * * *        1-8,9-16,17-24,25-29 2004
2004 * * * 1:8 * *        every 8 days in 2004, ⌈366/8⌉ ds

2004/7/1:2005/3/15 * * * 1:8 * *

                          every 8 days from 2004/7/1 to 2005/3/15
```

Source specification:

```
$URI     http://aurapar2u.ecs.nasa.gov
$DIR     airspar1/Aqua_AIRS_Level2/AIRX2STC.005
$FILE    $Y/$DY/AIRS\.$Y\.$M\.$D\..*\.hdf
or
$FILE    $Y/$DY/.*\.hdf
```

5.2 Callback Agent

When invoked, the callback agent determines if enough data have been ingested by the DIS to satisfy the callback specification in the Master List for the source. If not, the agent exits; another callback agent will be scheduled on the next download of data from the source. Otherwise, the callback agent marks the formatted data busy and invokes the staging script; when the script is finished, the agent exits, marking the data as completed.

5.3 Staging Script

The staging script is responsible for running the Data Assimilation, Surveillance, and Visualization components of the CO_2DAS. The staging script has access to the DIS data directories containing the formatted and transformed data, and invokes each component in turn, storing intermediate results in the DAS. When finished, the staging script exits.

Space Management

Whenever an acquisition or callback agent runs, it first determines the amount of free space in the DAS; if below a specified threshold, it schedules a garbage collection agent, and re-schedules itself for several hours into the future.

A GC agent looks through the Master List and deletes cached DAS data marked completed, in LRU order, until the amount of free space rises above a specified threshold, and logs the event.

Resources

UM CAC

TeraGrid