**MAS Ingestion Process**

* Setup of server, database and datasets -

By

Arapaut V. Sivaprasad

15 November 2018

**FOREWORD**

This document describes the steps involved in setting up datasets for services such as GEOGLAM. It is intended to be as complete as possible from start to finish, such as setting up the VM, creating database, crawling and ingesting the data. You need only the car key to drive, but the car manual is required to change a fuse. Similarly, this document is required only if you run into trouble. The component programs have been streamlined so that one or a few simple “key” commands will setup the system.

**DISCLAIMER**

This is an evolving document and may go out of sync with the programs in time. Those who maintain the programs are requested to update this document as well. While every care is taken to be factually and operationally correct including the directory paths, filenames, etc., there may be instances where certain aspects are not described correctly or there may be reasons for some steps to be different.

# TL;DR

* The script to crawl and ingest the data is ‘/…/gsky/crawl/[**crawl.sh**](https://github.com/asivapra/gsky/blob/master/crawl/crawl.sh)’
  + Edit it to insert the correct pathnames as e.g.
    - export **CRAWL\_DIR**=/g/data2/tc43/modis-fc/v310/tiles/8-day/cover/
    - export **CRAWL\_OUTPUT\_DIR**=/home/900/avs900/crawl\_outputs
  + Execute it from a shell command as ‘**sudo ./crawl.sh**’
* Setting up the VM is described [here](https://github.com/asivapra/gsky/blob/master/install/README.md).

# BACKGROUND

The crawler processes the **\*.nc** files to take the metadata and inserts it into several database tables. The table, ‘ingest’, is a transient path for the data while it is being formatted and put into other tables. There are three columns in this table as given below and two triggers. These triggers call other functions that format the data and insert into appropriate tables. This process is explained in detail in SECTION II.

Unlogged table "public.ingest"

Column | Type | Collation | Nullable | Default

---------+-------+-----------+----------+---------

in\_path | text | | |

in\_type | text | | |

in\_json | jsonb | | |

Triggers:

ingest BEFORE INSERT ON ingest FOR EACH ROW EXECUTE PROCEDURE ingest\_line()ingested AFTER INSERT ON ingest FOR EACH STATEMENT EXECUTE PROCEDURE ingested\_lines()

The ‘in\_json’ column records the details that can be overlaid as images on GEOGLAM web site.

The entire process of crawling and recording the data is described in this document.

# SECTION I

Descriptions of the original scripts are given in this section. Enhancements in terms of combining the scripts and increasing the efficiency are briefly described.

# SECTION II

For the crawling to work, there needs to be some pre-requisite steps such as creating the databases, functions and tables. These are done with four \*.sql files. Details of the \*.sql files used in the setup of databases and environments are described in this section. Included are also descriptions of the database tables and functions.

# SECTION III

Consolidated script to run the entire process in one command.

# SECTION I

## PROCESS FLOW

The crawling process is done through a series of bash scripts as shown below.

**Fig.1**

crawl\_pipeline.sh

ingest\_pipeline.sh

crawl.sh

shard\_ingest.sh

ingest.sh

shard\_create.sh

shard\_refresh.sh

## SHORT DESCRIPTION

* ‘crawl.sh’ sets up environment variables and calls ‘crawl\_pipeline.sh’
* ‘crawl\_pipeline.sh’ reads the \*.nc files (using GDAL) to create a TSV file.
* ‘crawl.sh’ then calls ‘ingest\_pipeline.sh’ to ingest the data in the TSV into database.
* ‘ingest\_pipeline.sh’ calls ‘shard\_create.sh’ to create the required PSQL tables and functions.
* ‘ingest\_pipeline.sh’ passes the TSV filepath to ‘shard\_ingest.sh’.
* ‘shard\_ingest.sh’ calls ‘ingest.sh’ to ingest the data from the TSV file.
* ‘shard\_refresh.sh’ refreshes the tables and functions.

## DETAILS OF SCRIPTS

### crawl.sh

This script sets up various environment variables to be used by the scripts that follow.

* export PATH="/local/gsky/bin:/local/gsky/share/mas:/local/gsky/share/gsky:$PATH"
  + This is required to find executables
* export CRAWL\_DIR=/g/data2/tc43/modis-fc/v310/tiles/8-day/cover/
  + This is the absolute directory path for the \*.nc files.
  + There can be sub-dirs in it, which will also be crawled.
  + To specify a few files, instead of a whole directory, define the variable, ‘CRAWL\_FILE\_LIST’. This, however, was not in the original ‘crawl.sh’, and was probably an error.
* export CRAWL\_OUTPUT\_DIR=/home/900/avs900/crawl\_outputs
  + This is where the TSV file is created.
  + If not given a ‘CRAWL\_FILE\_LIST’, a list of files will be created in this directory.
* export CRAWL\_CONC\_LIMIT=2
  + This is to control how many parallel processes must be run simultaneously.
  + The default is 16, but since the VM often has 1 to 8 CPUs, specify a number here.
    - Using a value twice as the ‘n\_cpus’ seems to work best.
* export LD\_LIBRARY\_PATH="/usr/local/lib:${LD\_LIBRARY\_PATH:-}"
  + Define the path where the libraries required by various programs.
* export PGUSER=postgres
  + This is to ensure that PSQL can be executed as user, ‘postgres’
  + The ‘root’ or any other user cannot run PostgreSQL, for security reasons.
* export PGDATA=/usr/local/pgsql/data
  + This is where the PostgreSQL data files reside.
* export GPATH=/g/data2/tc43
  + It is the base directory for the \*.nc files. It gets added to ‘public.shards’ to denote the dir\_path for the ‘shard’ or schema.

The script then does the following before calling ‘crawl\_pipeline.sh’

* Check whether \*.nc files are present in the ‘$CRAWL\_DIR’
  + Exit if none.
* Create the $CRAWL\_OUTPUT\_DIR
* Run the ‘crawl\_pipeline.sh’
* Run the ‘ingest\_pipeline.sh’
* Exit

### crawl\_pipeline.sh

This script checks the defined environment variables, sets up a file list of the \*.nc files and calls ‘GDAL’ via an executable, ‘gsky-crawl’, to read each \*.nc file and add its data to a TSV file. This TSV file is the one used by the ingest script that follows.

* Check whether $CRAWL\_OUTPUT\_DIR exists and, if not, create as a sub-dir of the script dir.
  + here="$( cd "$( dirname "${BASH\_SOURCE[0]}" )" && pwd )"
  + data\_dir=$here/crawl\_tsv/`date +'%Y-%m-%d\_%H-%M-%S'`
* Check if $CRAWL\_FILE\_LIST is defined.
  + If yes, create a ‘$file\_list’ with its values.
  + If not, get a listing of the \*.nc files in $CRAWL\_DIR to make the $file\_list.
    - job\_id="${find\_dir//[\/]/\_}"
    - file\_list=$data\_dir/${job\_id}.filelist
* Stream the names of files in $file\_list to ‘gsky-crawl’ to create the TSV file
  + cat $file\_list | concurrent -i -l $conc\_limit xargs bash -c 'gdal\_json "$@"' \_ | gzip > $crawl\_file
  + The above runs ‘'gdal\_json’ in parallel on $conc\_limit of CPUs
    - in turn calls ‘gsky-crawl’
  + The output is added to $crawl\_file and zipped.
    - I think it is wrong to gzip at this stage, as the text file is required later.
    - I think it must be kept as plain TSV file, used later and deleted.

### ingest\_pipeline.sh

This script takes the data stored in the TSV file (see previous section) and add it to the database. Though this script looks long and complex, all it does is setting up the environment for ingesting the data and then calling another script to ingest it. There are some functions in this script that apparently can be eliminated by simply defining one variable, ‘gpath’.

Check whether the environment variable, $GPATH, has been set.

* + If not, then construct the variable, $gpath, from $CRAWL\_DIR
    - Can eliminate two functions, get\_gpath() and assert\_gpath() (62 lines), by defining $GPATH
* Calls ‘shard\_create.sh’ to create a database schema.
* Calls ‘shard\_ingest.sh’, which in turn calls ‘ingest.sh’ to ingest the data.
  + Into the ‘mas’ database table, ‘ingest’.
* Calls ‘shard\_refresh.sh’
  + Purpose is unknown!

### shard\_create.sh

The primary purpose of this script is just to create a schema named “$shard”. Before creating the schema, it does some checking to see if it already exists.

### shard\_ingest.sh

This script just calls another, ‘ingest.sh’, and is puzzling why the latter cannot be called directly from the ‘ingest\_pipeline.sh’.

### ingest.sh

This script reads the TSV file and adds the data into a database tables. The mas:ingest is the table into which the data is ingested, but nothing gets added to it. Instead, the triggers on this table call other functions that re-format and add the data to other tables (see below).

### shard\_refresh.sh

This script runs functions that update some views and tables.

* select refresh\_views();
* select refresh\_polygons();
* select refresh\_caches();

## OUTCOME

Upon successful completion, the following tables and views in the ‘mas’ database will be populated.

### Tables:

#### metadata

* md\_hash | md\_ingested | md\_type | md\_json
* e.g.
  + bd788deb-fa0d-ebea-1dac-a2c5999a364c
  + 2018-11-21 15:01:34.395869+11
  + gdal
  + {"filename": "/g/data2/tc43…”… -1]}]}

#### paths

* pa\_hash | pa\_ingested | pa\_type | pa\_path | pa\_parents
* e.g.
  + e2a71936-e060-3431-e0af-c4c1e0d7a926
  + 2018-11-22 13:40:47.980522+11
  + Null
  + /g/data2/tc43/modis-fc/v310/tiles/8-day/cover/FC.v310.MCD43A4.h18v04.2018.006.nc
  + {399ab314-4e5e-e928-7ec4-94b96feb2d3f,d835f4d1-b7a9-9857-5fe4-166118e91ded,…}

#### tallies

* ta\_hash | ta\_count | ta\_size
* e.g.
  + 055ec5de-6554-e594-8a5c-e13507336d04 | 5 | 4096

### Materialized views:

#### directories

* di\_hash | di\_parent | di\_name | di\_ctime | di\_mtime | di\_atime | di\_mode | di\_inode | di\_uid | di\_gid | di\_user | di\_group | di\_count | di\_size
* e.g.
  + 399ab…b2d3f | | g | | | | | | | | | | 0 | 0
  + d835f…91ded | 399af…b2d3f | data2 | | | | | | | | | | 11 | 28672
  + de66a…bac77 | d835f…91ded | tc43 | | | | | | | | | | 10 | 24576
  + 34165…9580b | de66a…bac77 | modis-fc | | | | | | | | | | 9 | 20480
  + 8e3a9…70ca5 | 34165…9580b | v310 | | | | | | | | | | 8 | 16384
  + 54228…f4848 | 8e3a9…70ca5 | tiles | | | | | | | | | | 7 | 12288
  + 3bd50…ba59b | 54228…f4848 | 8-day | | | | | | | | | | 6 | 8192
  + 055ec…36d04 | 3bd50…ba59b | cover | | | | | | | | | | 5 | 4096

#### polygons

* po\_hash | po\_stamps | po\_min\_stamp | po\_max\_stamp | po\_name | po\_pixel\_x | po\_pixel\_y | po\_polygon
* e.g.
  + ed289005-7578-3180-3b86-8b20a3d9c755 | {"2017-01-01 11:00:00+11”,…} | 2017-01-01 11:00:00+11 | 2017-12-27 11:00:00+11 | bare\_soil | 463.50584396298467 | -463.50584396298467 | 0103000…9734941

The TSV file appears to be transient and is probably not required to be kept. Another transient file, “\*.filelist” is also not required to be kept, but both these are being retained.

## ENHANCEMENTS

### Consolidation of the scripts

Instead of having 7 separate scripts, combining them all into one is feasible. See below. There may be reasons for having them as separate scripts, though.

crawl\_and\_ingest.sh

**Fig.2**

The above combined step can also be split as below. It is to enable the use of ingesting script separately from the crawling script.

**Fig.3**

ingest.sh

crawl.sh

### Speeding up the execution

The crawling phase is the most time-consuming part. To crawl 4,896 \*.nc files it took 64 minutes on an 8CPU virtual machine, even though the ‘gsky-crawl’ is run concurrently (limit=2). Increasing the concurrency to 8 (i.e. 1x CPUs) reduced the time to 20 minutes and concurrency to 32 (i.e. 4x CPUs) reduced it to 14 min. The server load goes up considerably (from 2 to 10 to 40) by doing so. Unless the VM is also being used for other purposes, even a load of 40 will not have any adverse effect. Increasing it even further may cause thrashing, though on regular servers a load of up to 100 is tolerable. On a production server it may slow down things for others. Perhaps a way is to do the crawling on a separate machine, ftp the TSV file across to the production server and ingest it there.

While the time is not excessive for small data sets like the one tried, it may become a limiting factor with larger data sets or with composite data files.

By using ‘parallel’ instead of ‘concurrent’ to run the processes it may be possible to speed up considerably. Parallel is not the same as ‘concurrent’ and will use up the available CPUs more efficiently. Running in batch mode will also improve the times, as the error lines displayed by GDAL are slowing down the I/O and adding to the total execution times.

# SECTION II

To setup the databases, tables and functions there are four \*.sql files used. Of these, three must be run once immediately after the VM is setup and the fourth (‘shard.sql’) is used each time during the ingestion process.

## [schema.sql](https://github.com/nci/gsky/blob/master/mas/db/schema.sql)

This is required to be run just once on a new VM. Only the steps relevant to ingestion are described below. The full code shall be viewed in the above file.

1. Kick off every other user, if any, connected to the database, ‘mas’.

update pg\_database set datallowconn = 'false' where datname = 'mas';  
select pg\_terminate\_backend(pid) from pg\_stat\_activity where datname = 'mas';

1. Delete the database, ‘mas’, and set it up again.

drop database if exists mas;  
…  
create database mas owner mas;

1. Execute another SQL file (see below for details)

\i util.sql

1. Create some required tables, functions and views.

create table shards ();

create table nci\_spatial\_ref\_sys ();

create function create\_views()

create materialized view public.paths\_common as…  
create view public.paths as…  
create materialized view public.directories\_common as…   
create view public.directories as…  
create view public.files as…  
create view public.links as…  
create view public.polygons as…  
create view public.polygon\_srids as…  
create view public.paths as…  
create view public.metadata as…  
create view public.netcdf as…  
create function refresh\_views()  
create function parent\_paths(dir text)  
create function path\_unhash(hash uuid)  
create function path\_hash(path text)  
create function path\_absolute(path text, relative text)

\echo Done!

[util.sql](https://github.com/nci/gsky/blob/master/mas/db/util.sql)

This sets up general utilities for PostgreSQL. It is called from within ‘scheme.sql’. The following functions are created. Follow the link for the above file for details of code.

create or replace function relation\_type()  
create or replace function drop\_functions()  
create or replace function age\_months()  
create or replace function generate\_month\_series()  
create or replace function is\_current\_year()  
create or replace function is\_current\_month()  
create or replace function is\_current\_day()  
create or replace function try\_json()  
create or replace function try\_integer()  
create or replace function try\_inet()  
create or replace function try\_date()  
create or replace function try\_timestamp()  
create or replace function try\_timestamptz()  
create or replace function notnull()