SCALABLE DATA PROCESSING WITH THE LROC PROCESSING PIPELINES.

K. N. Paris, N. M. Estes, E. Cisneros, and M. S. Robinson, School of Earth and Space Exploration, Arizona State University, Tempe, AZ 85287.

Introduction: The Lunar Reconnaissance Orbiter Camera (LROC) is a suite of three cameras onboard the Lunar Reconnaissance Orbiter (LRO), which has been systematically mapping the Moon since 2009 [1]. The LROC includes two Narrow Angle Cameras (NACs) and one Wide Angle Camera (WAC) [2] that collect hundreds of images each day, totaling over 2.2 million observations as of November 2017.

In addition to the science files, the LROC Science Operations Center (SOC) generates and receives from the Mission Operations Center (MOC) almost 100 other types of files. Many of the file types are necessary to plan observations, facilitate science file processing, and generate products for the scientific community and the public. Keeping track of these diverse file types requires a well thought out and scalable process for future expansion or compression: additional file types, additional processing steps to pipelines, retiring pipelines, and future mission support.

Processing Strategy: There are four components that have made the processing strategy for the LROC SOC as successful as it is. These are 1) the processing pipelines, 2) a mission database, 3) a job and resource management software suite, and 4) a robust file system.

Pipeline Scripts aka Procedures: Individual scripts are referred to as "procedures" and collections of procedures relating to a particular file type are referred to as "pipelines". Each procedure is written to accomplish a very specific task and can chain to other procedures as necessary. This approach was taken to keep the scripts uncomplicated for the ease of development and reprocessing purposes, and to be able to simplify error handling (reduce duplicate processing or rolling back any processing performed).

The procedures are written in Bash. Bash was chosen as the preferred language for read-ability by the Operations staff (who aren't necessarily software developers), who would be creating or updating the pipelines. In some of the Bash scripts, there are calls to programs that are written in C++ or Ruby where appropriate (i.e. image processing or geometry calculations).

Whenever a file is received or generated by the LROC SOC it is "ingested" – this is the process by which the file and ancillary data are cataloged in a database. After a file is ingested, it is passed to the "pipeline" for that file type where further processing occurs via individual procedures.

The pipelines and procedures are named in a way that makes it easy to identify which file type that pipeline processes and the order in which procedures are to be executed. For example (Fig. 1): lroc_moc27_p30 is the third step (p30) in processing of the moc27 file type.

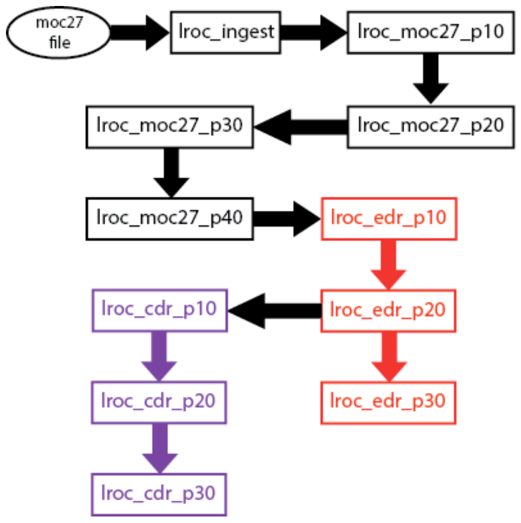

Figure 1: Example of the processing flow for a moc27 file type through the moc27 pipeline (black), EDR pipeline (red), and CDR pipeline (purple).

Figure 1 is an example of a file being processed through the several relevant pipelines. A science file is received from the MOC (file type moc27) and submitted to the ingest script where it is initially cataloged It is then passed to the first step in the moc27 processing pipeline which includes four procedures (lroc_moc27_p10 – p40).

The first procedure compares the values in the science file header to the expected values that are stored in the database, and creates and updates fields in the database to indicate that a science file for an observation has been received. The p10 procedure then submits the science file to the second (p20) procedure, which updates specific timing-related fields in the database for the observation and passes the file onto the p30 script. The third procedure (p30) checks the file for validity and computes statistics from the image data and sends the file to the final procedure in the moc27 pipeline (p40). The fourth procedure (p40) reviews the image statistics, checks for housekeeping and SPICE data and computes the data quality id for the observation. Assuming that all of the necessary data are present for the science file, the lroc_moc27_p40 procedure then submits the science file to the EDR pipeline, which starts with the lroc_edr_p10 procedure. Assuming success in the p10 and p20 procedures, the file is submitted to the final procedure in the EDR pipeline and also to the first procedure in the CDR pipeline.

Whenever a job fails, the user is made aware of it via the Rector interface [3]. The output of the script is saved for review and mitigation development. Once the necessary mitigation steps have been taken for successful processing, the job is resubmitted to the pipeline that it failed out of and processing can proceed.

Database: The LROC SOC uses a PostgreSQL database to track received files and track the status of the processing for the files. A record is created for each file received and meta-data captured from the file. Most commonly, this meta-data consists of the start and end times of a file, the delivery time, file md5 checksum and file path in the file system.

Resource Management. Unless paused, the pipelines are always "on" and available to process whenever jobs are submitted. This requires careful resource management on the processing cluster, which is done by an in-house developed program called Rector. More information about the Rector software can be found in the "Taming Pipelines, Users, and High Performance Computing with Rector" [3].

File System: The LROC SOC maintains a fully redundant 2.3 Petabyte storage array where project files are organized and stored. The storage array is shared via the NFS protocol to our SOC computer cluster, as well as mission workstations to facilitate sharing LROC files. Cluster nodes make use of locally attached disks for pipeline processing. If new file products are generated, that pipeline step will move the file product to the shared storage array.

Development and Testing: The LROC pipelines are under version control using Subversion control software [4]. Any script that is related to the pipelines and the pipelines themselves are stored in the same subversion project. When updates are made to any of the software, it is deployed to a separate test cluster, integration testing performed, and when successful, committed to the subversion project. Once testing is successful and the changes committed, a new release is tagged, the affected pipelines are paused by the Operations staff, and the newly tagged subversion project is deployed to the production file system.

Evolution of the Pipelines: The pipelines started off as a simple set of scripts for ingesting and processing files as they are received. As the mission continued and operations evolved the pipelines have expanded into a set of 132 procedures to handle new file types requested from the MOC, some error handling situations could be handled automatically, and added steps to increase the usability of the data for the greater science community. Several key lessons learned during the evolution process are described in detailed below.

Timing Challenges: Given the uncertainty in timing of file delivery and the dependencies of some file types on others, one of the biggest challenges faced was how to process files in the right order when they aren't necessarily received in an required order. For example, processing a science file requires SPICE files and housekeeping telemetry files which may come before or after the science file itself.

To mitigate the problem of file delivery timing, meta-data for each file are stored in the database, including file type for each file and the start and end times for files where those data exist. A separate non-pipeline script was written and uses this meta-data to determine if given file types exists for a particular observation or science file. This script is called by each pipeline for which there is a file processing dependency. In the example of the science file processing, the script is called in the science file pipeline, SPICE file pipelines, and in the housekeeping pipeline. This ensures that no matter what the delivery times of the files are, the science file can be processed automatically.

Keep multiple instances of a job from running: In situations when one job is resource-intensive, multiple instances that job running at the same time was be problematic. This would cause computers on the cluster or the production database to greatly slow down, which affects the Operations staff and anyone else utilizing the processing cluster. To deal with this, lock files were implemented so that when one instance of a job starts, it checks for a lock file before starting the intensive work and if a lock file exists, the job waits and checks for the lock file's existence at set intervals (i.e. 30-60 seconds). If a lock file does not exist, the script creates a lock file to prevent other instances of the job from running.

Reprocessing: Files need to be reprocessed as calibration is updated, geometry is improved, or for any other myriad of reasons. The pipelines were written in such a way that jobs can be submitted to a single procedure to kick-off reprocessing of particular files or file types.

Future work: Future work will focus on transferring knowledge and processes to new missions currently in development, as well as continued refinement for the LRO mission.

References: [1] Chin et al., Space Sci Rev (special issue. [2] Robinson et al., Space Sci Rev. [3] Estes et al., submitted this volume. [4] https://subversion.apache.org/