Taming Pipelines, Users, and High Performance Computing with Rector. N. M. Estes, K. S. Bowley, K. N. Paris, V. H. Silva, M. S. Robinson, School of Earth and Space Exploration, Arizona State University

Introduction: The Lunar Reconnaissance Orbiter Camera (LROC) Science Operations Center (SOC) receives and processes ~450 gigabits of data every day. These data get uncompressed and processed into engineering data records (EDRs) and calibrated data records (CDRs). During processing, there are many steps to catalog, calibrate, calculate geometry, validate end products, and many other steps (132 pipeline procedures in all), and it requires a well-designed system to orchestrate all of these steps for thousands of products each week. In addition to this baseline processing, users require a wide variety of tasks to be run on thousands of images for map projecting, mosaicking, photometric correction, and other tasks. This processing is currently coordinated by Rector over a 14-node processing cluster running a total of 634 CPU cores.

The baseline requirements for the job management system included the ability to automatically allocate CPUs and RAM on a processing node, coordinate between hundreds of CPU cores over many nodes, provide a GUI interface for operations staff to monitor processing and handle exceptions, and provide a way for users to distribute arbitrary jobs across the cluster without specialized knowledge. Before creating Rector, the LROC SOC evaluated several job control systems including HiRISE Conductor [1], Condor [2], and Torque [3]. While these options handled some of the requirements, no single solution met all requirements. After developing an initial prototype, the team decided to develop a job control system that met all of the needs of the LROC SOC.

Job Management: Rector handles two categories of jobs: automated pipeline jobs and user jobs. These two categories are handled differently to meet the needs of the users of each job type. In both cases, the job queues are managed via a central PostgreSQL database using row exclusive locks to ensure that each job runs only once. Logs of every job run by Rector are also kept in the database with all information necessary to re-run the job in the future if necessary.

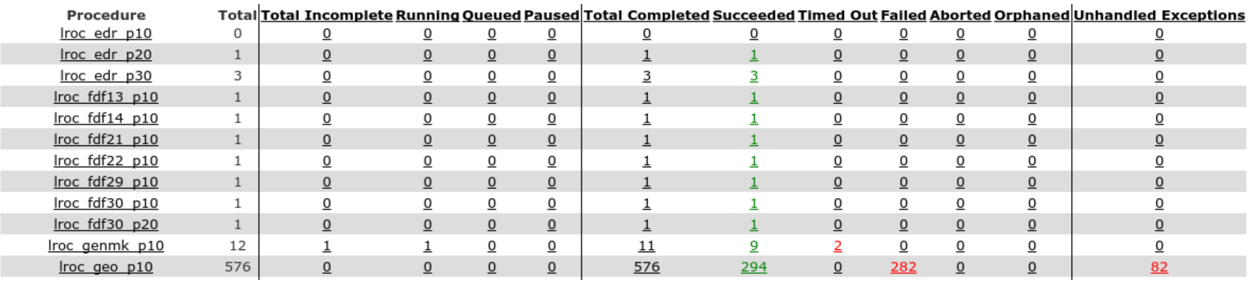
The GUI interface for LROC SOC operations staff is written using the Ruby on Rails framework. The Rector daemon that runs on each processing node, command line tools for job management, and administrative tools are written in Ruby.

Pipeline Job Management: Pipeline jobs are automated procedures that ingest all files received from the mission operation center and perform all necessary processing steps to generate EDRs, CDRs, browse products, histograms; as well as provide quality control, statistics calculation, and many other tasks necessary for LROC operations. These jobs are managed through a GUI interface (Figure 1) where the LROC SOC operations staff can see all job status information and handle any exceptions that may have occurred. This interface provides a general overview of every node in the cluster and every configured pipeline procedure, along with the ability to drill down to the fine details if necessary (Figures 2, 3).

The pipeline procedure configuration allows for procedures to be prioritized (for example, importing SPICE kernels is a higher priority than producing CDR products). Operations staff can further control how things are run by specifying a regular expression that gets matched against the procedure name to control which procedures are allowed to run on a given node.

Details on how the LROC pipelines work can be found in the "Scalable Data Processing with the LROC Processing Pipelines" abstract also submitted to this conference.

User Job Management & Security: User jobs are handled differently than pipeline jobs in Rector to allow arbitrary commands to be submitted to the cluster. The command line tools for managing user jobs were modeled after the high performance computing standard OpenPBS command line tools [4]. User jobs run at a lower priority than pipeline jobs, and when running user jobs, Rector attempts to balance jobs submitted by different users as much as possible. For example, if user A is using all available CPUs in the cluster and user B submits a set of jobs, as jobs finish, Rector



Procedure	Total	Total Incomplete	Running	Queued	Paused	Total Completed	Succeeded	Timed Out	Failed	Aborted	Orphaned	Unhandled Exceptions
lroc_edr_p10	0	0	0	0	0	0	0	0	0	0	0	0
lroc_edr_p20	1	0	0	0	0	1	1	0	0	0	0	0
lroc_edr_p30	3	0	0	0	0	3	3	0	0	0	0	0
lroc_fdf13_p10	1	0	0	0	0	1	1	0	0	0	0	0
lroc_fdf14_p10	1	0	0	0	0	1	1	0	0	0	0	0
lroc_fdf21_p10	1	0	0	0	0	1	1	0	0	0	0	0
lroc_fdf22_p10	1	0	0	0	0	1	1	0	0	0	0	0
lroc_fdf29_p10	1	0	0	0	0	1	1	0	0	0	0	0
lroc_fdf30_p10	1	0	0	0	0	1	1	0	0	0	0	0
lroc_fdf30_p20	1	0	0	0	0	1	1	0	0	0	0	0
lroc_genmk_p10	12	1	1	0	0	11	9	2	0	0	0	0
lroc_geo_p10	576	0	0	0	0	576	294	0	282	0	0	82

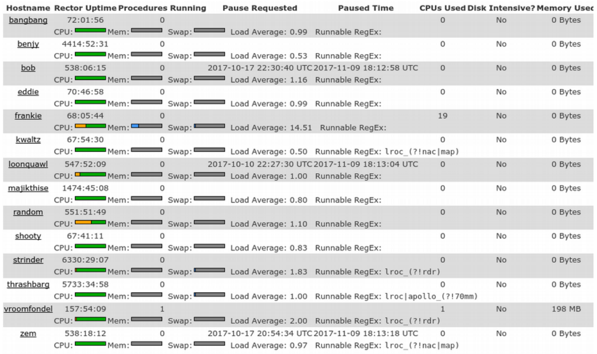
Figure 1: Small selection of pipeline procedures showing current procedure queue status as well as log status over a user-defined time interval.

will start user B's jobs in preference until the number of CPUs used by user A and user B match, at which time it will alternate between starting jobs from user A and user B to maintain that balance. Likewise, if user C also submits jobs, user C's jobs will be selected in preference until a balance between all three users is achieved. Rector will never stop a running job. Prioritization and user-balancing only occur when Rector selects a new job to run.

For security purposes, user jobs are run as the user that submitted the job. This way the user's normal permissions and access restrictions also apply to the jobs they submit to Rector. Because these jobs are queued in a central database and the user needs to be able to access the database to add jobs to the queue, a security mechanism is in place to ensure that users can only submit jobs that run as themselves and that submitted commands cannot be modified. To achieve this, each user has a public/private key-pair for Rector. This private key should be protected the same as any SSH, PGP, or other private key. When submitting a job, the job submission tool generates a cryptographic signature of the command to be run using the user's private key, and this signature is stored in the database. When Rector selects a user job to run, it loads the user's public key and verifies that the signature matches the command to run. After the initial setup of the key, this process is completely transparent to the user, but if a signature does not match a command, Rector will generate an error and refuse to run the job.

Results: To date, Rector has shepherded over 4.3 million EDR and CDR products through the LROC pipeline. Between those products, and all the other pipeline procedures, Rector has run over 71.2 million pipeline jobs since 2009. In addition to the pipeline jobs, Rector has also been used to handle more than 40.1 million user jobs in that time.

Rector has proven to be a reliable and easy-to-use job management tool at the LROC SOC. It has grown from an initial processing cluster of 6 nodes and 12 CPU cores to the current cluster size of 14 nodes and 634 CPU cores. Rector's prioritization capabilities allow the LROC SOC operations staff to manage how jobs are run when necessary, but most of the time, Rector's own ability to manage resources handles all job marshaling in a completely hands-off manner. The job logging allows operations staff to look back on processing history at any time, and job errors are presented to operations staff in an intuitive way that allows human intervention when necessary.

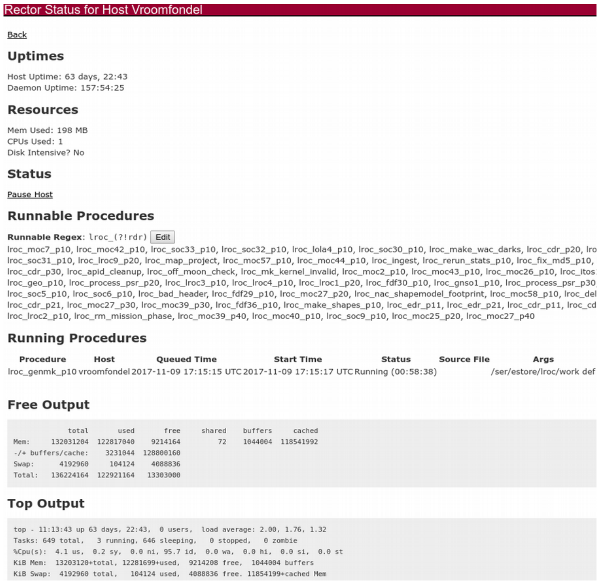


Hostname	Rector Uptime	Procedures Running	Pause Requested	Paused Time	CPUs Used	Disk Intensive?	Memory Usec
bangbang	72:01:56	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.99	Runnable RegEx:			
benjy	4414:52:31	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.53	Runnable RegEx:			
bob	538:06:15	0	2017-10-17 22:30:40 UTC	2017-11-09 18:12:58 UTC	0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 1.16	Runnable RegEx:			
eddie	70:46:58	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.99	Runnable RegEx:			
frankie	68:05:44	0			19	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 14.51	Runnable RegEx:			
kwaltz	67:54:30	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.50	Runnable RegEx: lroc_(?!nac|map)			
loonquawl	547:52:09	0	2017-10-10 22:27:30 UTC	2017-11-09 18:13:04 UTC	0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 1.00	Runnable RegEx:			
majikthise	1474:45:08	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.80	Runnable RegEx:			
random	551:51:49	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 1.10	Runnable RegEx:			
shooty	67:41:11	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.83	Runnable RegEx:			
strinder	6330:29:07	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 1.83	Runnable RegEx: lroc_(?!rdr)			
thrashbarg	5733:34:58	0			0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 1.00	Runnable RegEx: lroc|apollo_(?!70mm)			
vroomfondel	157:54:09	1			1	No	198 MB
	CPU:	Mem:	Swap: Load Average: 2.00	Runnable RegEx: lroc_(?!rdr)			
zem	538:18:12	0	2017-10-17 20:54:34 UTC	2017-11-09 18:13:18 UTC	0	No	0 Bytes
	CPU:	Mem:	Swap: Load Average: 0.97	Runnable RegEx: lroc_(?!nac|map)			

Figure 2: List of nodes in the processing cluster with summary information on procedures running, CPU cores used, and other diagnostic information.

Future Work: Rector currently has a limitation when hundreds of short-duration jobs are submitted to a cluster with a large number of CPU cores. Rector will successfully run the jobs, but it will not use all available cores in that case due to the timing of the exclusive lock necessary in the database to ensure that the Rector daemons coordinate successfully between themselves. To avoid this issue, it is recommended that users avoid submitting jobs that run in less than 30 seconds. It is suggested that these short-duration jobs be batched-up in chunks that take longer than 30 seconds, so that Rector is able to keep all cores in the cluster running at full capacity. Work to eliminate that inconvenience would be helpful given enough time in the development schedule.

References: [1] Schaller, C.~J.\ 2006, 37th Annual Lunar and Planetary Science Conference, 37 [2] Douglas Thain, Todd Tannenbaum, and Miron Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005. [3] Torque http://www.adaptivecomputing.com/products/open-source/torque/ [4] OpenPBS http://www.mcs.anl.gov/research/projects/openpbs/



Rector Status for Host Vroomfondel

Back

Uptimes

Host Uptime: 63 days, 22:43
Daemon Uptime: 157:54:25

Resources

Mem Used: 198 MB
CPUs Used: 1
Disk Intensive? No

Status

Pause Host

Runnable Procedures

Runnable Regex: lroc_(?!rdr) Edit
lroc_moc7_p10, lroc_moc42_p10, lroc_soc33_p10, lroc_soc32_p10, lroc_lola4_p10, lroc_soc30_p10, lroc_make_wac_darks, lroc_cdr_p20, lro
lroc_soc31_p10, lroc_lroc9_p20, lroc_map_project, lroc_moc57_p10, lroc_moc44_p10, lroc_ingest, lroc_rerun_stats_p10, lroc_fix_md5_p10,
lroc_cdr_p30, lroc_apid_cleanup, lroc_off_moon_check, lroc_mk_kernel_invalid, lroc_moc2_p10, lroc_moc43_p10, lroc_moc26_p10, lroc_itos
lroc_geo_p10, lroc_process_psr_p20, lroc_lroc3_p10, lroc_lroc4_p10, lroc_lroc1_p20, lroc_fdf30_p10, lroc_gnso1_p10, lroc_process_psr_p30
lroc_soc5_p10, lroc_soc6_p10, lroc_bad_header, lroc_fdf29_p10, lroc_moc27_p20, lroc_nac_shapemodel_footprint, lroc_moc58_p10, lroc_del
lroc_cdr_p21, lroc_moc27_p30, lroc_moc39_p30, lroc_fdf36_p10, lroc_make_shapes_p10, lroc_edr_p11, lroc_edr_p21, lroc_cdr_p11, lroc_cd
lroc_lroc2_p10, lroc_rm_mission_phase, lroc_moc39_p40, lroc_moc40_p10, lroc_soc9_p10, lroc_moc25_p20, lroc_moc27_p40

Running Procedures

Procedure	Host	Queued Time	Start Time	Status	Source File	Args
lroc_genmk_p10	vroomfondel	2017-11-09 17:15:15 UTC	2017-11-09 17:15:17 UTC	Running (00:58:38)		/ser/estore/lroc/work def

Free Output

```
             total       used       free     shared    buffers     cached
Mem:     132031204  122817040    9214164         72    1044004  118541992
-/+ buffers/cache:    3231044  128800160
Swap:      4192960     104124    4088836
Total:   136224164  122921164   13303000
```

Top Output

```
top - 11:13:43 up 63 days, 22:43,  0 users,  load average: 2.00, 1.76, 1.32
Tasks: 649 total,   3 running, 646 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.1 us,  0.2 sy,  0.0 ni, 95.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  13203120+total, 12281699+used,  9214208 free,  1044004 buffers
KiB Swap:  4192960 total,   104124 used,  4088836 free. 11854199+cached Mem
```

Figure 3: Node detail page showing a variety of Rector daemon and host diagnostic information.