Using Resource Utilization Reporting to Collect DVS Usage Statistics

Tina Butler
National Energy Research Scientific Computing Center
Ernest Orlando Lawrence Berkeley National Laboratory
Berkeley, CA USA
Email: tlbutler@lbl.gov

Abstract--**In recent releases of the Cray Linux Environment, a feature called Resource Utilization Reporting (RUR) as been introduced. RUR is designed as an extensible framework for collecting usage and monitoring statistics from compute nodes on a per application basis. This paper will discuss the installation and configuration of RUR, and the design and implementation of a custom RUR plugin for collecting DVS client-side statistics on compute nodes.**

***Keywords*-RUR, DVS**

I. Cray Accounting: A Brief Recap

Beyond charging for usage, historically, accounting data has been useful for characterizing a system's workload. Memory usage, I/O load and patterns of communication are valuable for optimizing system use and applications, for provisioning or upgrading existing system, and for generating requirements for new systems.

In the days of vector machines, Cray provided a utility under UNICOS – Cray System Accounting (CSA) - that collected numerous job statistics including CPU time, memory high-water mark and averages, and block and character I/O counts. After Cray Research was acquired by Silicon Graphics, CSA was ported to Linux, renamed Comprehensive System Accounting and open-sourced. CSA is still supported by Cray for the Cray Linux Environment (CLE), but the current implementation has not scaled reliably with larger parallel systems, and some of the more useful features of previous versions are no longer available.

In response to customer requirements, in 2010, Cray added a new feature to CLE, Application Resource Utilization (ARU). ARU is integrated with the Application Level Placement Scheduler (ALPS) and provides basic usage statistics for each aprun. ARU data (obtained from the kernel via an rusage call) is incorporated on the apsys finishing line written to syslog or can be written to a flat file. (Figure 1.) One of the issues we discovered with ARU is that if a job terminates abnormally, ARU data is not reported. Many NERSC users submit batch jobs that terminate by hitting wallclock limit. Unfortunately, hitting wallclock limit is an error that causes ARU data to be lost. Also, ARU is not extensible by customer sites.

Figure 1. ARU Data in syslog

```
<150>1 2014-04-17T00:00:05.982308-07:00 c5-0c2s4n3 apsys 19438 p0-20140403t113614 [alps_msgs@34] apid=28108121, Finishing, user=56395, batch_id=7447167.hopque01, exit_code=0, exitcode_array=0, exitsignal_array=0, utime=521, stime=41, maxrss=1425528, inblocks=443257, outblocks=801443, cpus=24, start=Wed Apr 16 23:50:43 2014, stop=Thu Apr 17 00:00:05 2014, cmd=smoothing
```

II. Resource Utilization Reporting

Shortly after ARU was released, Cray was requested to expand the type and quantity of data collected and to make it site-customizable. As an alternative to augmenting ARU, Cray chose to start afresh and introduced Resource Utilization Reporting (RUR) in CLE 4.2 and 5.1.

RUR is designed as a scalable, extensible framework for collecting information from compute nodes. Like ARU, RUR is initiated by ALPS, but not tightly integrated. Cray provides the RUR framework and a set of plugins for data collection and staging, post processing, and output. Data plugins collect and stage data on compute nodes and post-process the data on MOM or login nodes, output plugins transfer the processed data to storage. [1]

RUR workflow is shown in Figure 2.

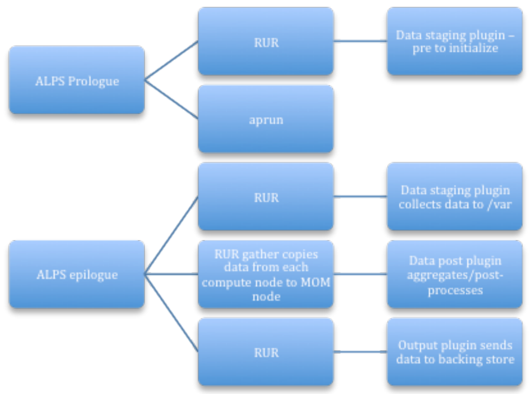
ALPS Prologue
RUR
Data staging plugin – pre to initialize
aprun
ALPS epilogue
RUR
Data staging plugin collects data to /var
RUR gather copies data from each compute node to MOM node
Data post plugin aggregates/post-processes
RUR
Output plugin sends data to backing store

Figure 2. ARU Workflow

Cray provides three sets of data plugins and three output plugins for RUR. The data plugins are:

- *taskstats* – this plugin provides basic process accounting information including user and system CPU time, memory usage and file input/output. This provides the data previously available from ARU.
- *gpustat* – this plugin provides utilization statistics for NVIDIA GPUs on XK and XC-30 systems, including compute time and memory usage.
- *energy* – this plugin records energy utilization for an application on XC-30 systems.

The output plugins are:

- *llm* – this plugin writes RUR output to the Lightweight Log Manager (LLM), which aggregates syslog information on the SMW or other site-defined syslog location.
- *file* – this plugin writes RUR output to a site-defined flat file.
- *User* – this plugin writes directly to a user directory if an environment variable is set. (CLE5.1 only)

RUR is installed by default in CLE, but not enabled. RUR is enabled by adding RUR information to the apsys stanza in either /etc/alps.conf (CLE 4.X) or /etc/opt/cray/alps/alps.conf (CLE 5.x). This includes the paths to the RUR prologue and epilogue and timeout values. The RUR configuration file is found in /etc/opt/cray/rur/rur.conf. This file has plugin definitions and values and allows the administrator to turn plugins on and off, set local values for file locations and define custom plugins to the RUR framework.

Please note that CLE must be configured with the /dsl environment for compute nodes as default as RUR relies on the service node shared root accessed via /dsl. CLE ships with /dsl as the default, but if that has been modified, the administrator must fall back to the default configuration.

III. The Need for a Custom Plugin

The National Energy Research Scientific Computing Center (NERSC) has an active and expanding interest in collecting many types of resource usage data in order to characterize the workload of our users. We are interested in collecting information on both a system-wide and per-application basis.

NERSC supports its users by providing them with access to a multi-petabyte, global parallel filesystem that is mounted on all the major compute platforms at NERSC. This is known as the NERSC Global Filesystem (NGF), and uses IBM's Global Parallel File System (GPFS).

NGF actually comprises multiple filesystem instances resident on different storage hardware platforms with different fabric connectivity (InfiniBand and 10 Gb Ethernet), different block sizes and different performance characteristics. The current set of NGF filesystems mounted by NERSC production systems is shown in Figure3.

/global/syscom, bs=65536
/global/common, bs=65536
/global/u1, bs=131072
/global/u2, bs=131072
/global/dna, bs=1048576
/global/project, bs=4194304, RDMA
/global/projectb, bs=1048576, RDMA
/global/scratch2, bs=8388608, RDMA

Figure 3. NERSC NGF filesystems with block sizes

On the Cray systems at NERSC, access to NGF filesystems from the compute nodes is through the Data Virtualization Service (DVS), which provides I/O forwarding service from XE and XC service nodes to the compute partition.

System-level usage information for I/O to filesystems is relatively easy to collect at the file servers; collecting per-application data is more difficult

In recent released versions of CLE, DVS statistics are available for both DVS servers and clients [2]. The system-level data on the DVS servers is available through the /proc filesystem and includes both IPC and request/operation statistics for a given server. The client data, also available through /proc, includes the same statistics per mount point. If we can collect those DVS client statistics from compute nodes it will allow us to begin to characterize NGF usage for individual applications. With the release of RUR, we now have a flexible mechanism for collecting and storing DVS client statistics.

IV. IMPLEMENTING A CUSTOM PLUGIN

The RUR framework is written in Python, but custom plugins are not required to be in the same language. The interfaces to the framework and functional requirements for plugins are defined and described in the Cray manual, 'Managing System Software for the Cray Linux Environment'. The data_staging plugin is called before the application runs, allowing counter initialization, and after the application completes to collect the desired data. Data is written to a file in /var/spool on the compute node and copied by the framework to a MOM or login node using pcmd. The data post plugin runs on the MOM/login node, and aggregates the data files copied from the compute nodes.

On each client compute node, DVS statistics are written to two areas in /proc/fs/dvs:

- /proc/fs/dvs/mounts/*/{mount,stats, openfiles) – the stats file has the per-mount point counts of successful and failed file system operations); the mount file has the mount point data including local and remote paths (Figures 4 & 5).
- /proc/fs/dvs/ipc/stats – this stats file has DVS IPC statistics including bytes transferred and received, and message counts. This file does not segregate data by mount point.

The dvs data staging plugin first resets the stats counters by writing a '2' into the stats file for each mount point in the pre-application phase. After the application completes, the plugin walks the /proc/fs/dvs/mounts sub-tree and collects the mount point and stats data. Statistics are written to /var/spool/RUR/dvs.*apid* on each compute node. The dvs post plugin aggregates the DVS statistics from each compute node and passes the data to the RUR framework for processing by the active output plugins.

RUR output from the dvs plugin is shown in Figure 6.

I'd like to make a few observations on implementing a custom plugin for RUR. First, the RUR framework for gathering and output expect a single line file for each node. This constrains custom plugins that might want to write output to a different backing store like a database or a set of file that are not aggregated. Second, it would be valuable to be able to explicitly tie data and output plugins to each other. Currently in the RUR framework, the products of all active data plugins will be passed to all active output plugins. This constrains the format of the data plugin to comply with the most restrictive output plugin. Last, it is important to know that error logging from RUR does not go to syslog, but is available in /var/log/apsys on the MOM node where the particular aprun was scheduled.

V. FURTHER WORK

The current dvs plugins are only collecting per-mount point filesystem operations data. The DVS IPC data is also of interest, the plugin set needs to be extended to incorporate at least part of that data. NERSC job statistics are aggregated in our "Job Completion" database; DVS client data collected by RUR will become part of that as well.

RUR is a valuable new capability with the potential for gathering a broad range of compute node –based data and a welcome addition to the administrative toolkit.

VI. ACKNOWLEDGEMENTS

This work was supported by the Director, Office of Science, Office of Advance Scientific Computing Research of the U.S. Department of Energy under contract No. DEAC02-05CH11231

VII. REFERENCES:

[1] Introduction to Cray Data Virtualization Service, S-0005-51-1
[2] Managing System Software for the Cray Linux Environment, S-2393-4202

```
local-mount /global/project
remote-path /global/project
options
(rw,blksize=4194304,nodename=c3-0c0s4n0:c7-
2c2s6n3,nocache,nodatasync,noclosesync,retry,failover,userenv,clusterfs,k
illprocess,nobulk_rw,noatomic,nodeferopens,no_distribute_create_ops,no_
ro_cache,maxnodes=1,nnodes=2,magic=0x47504653)
active_nodes c3-0c0s4n0 c7-2c2s6n3
inactive_nodes
remote-magic 0x47504653
```

Figure 4. Example of /proc/fs/dvs/mounts/[0-x]/mount

```
RQ_LOOKUP: 8994092 0              RQ_OPEN: 68151 0
RQ_CLOSE: 68151 0                 RQ_READDIR: 23753 0
RQ_CREATE: 698 0                  RQ_UNLINK: 337 0
RQ_LSEEK: 0 0                     RQ_IOCTL: 0 0
RQ_FLUSH: 0 0                     RQ_RELEASE: 0 0
RQ_FSYNC: 0 0                     RQ_FASYNC: 0 0
RQ_LOCK: 0 0                      RQ_LINK: 0 0
RQ_SYMLINK: 2 0                   RQ_MKDIR: 12 0
RQ_RMDIR: 0 0                     RQ_MKNOD: 0 0
RQ_RENAME: 37 0                   RQ_READLINK: 27312 0
RQ_TRUNCATE: 6 0                  RQ_SETATTR: 2074 0
RQ_GETATTR: 313266 0              RQ_PARALLEL_READ: 19034471 0
RQ_PARALLEL_WRITE: 1408148 77RQ_STATFS: 11 0
RQ_READPAGE_ASYNC: 4555 0         RQ_READPAGE_DATA: 4555 0
RQ_GETEOI: 0 0                    RQ_INITFS: 0 0
RQ_SETXATTR: 236 0                RQ_GETXATTR: 49 0
RQ_LISTXATTR: 0 0                 RQ_REMOVEXATTR: 0 0
RQ_VERIFYFS: 0 0                  RQ_GET_LANE_INFO: 0 0
RQ_RO_CACHE_DISABLE: 0 0          RQ_PERMISSION: 5329 0
read_min_max: 0 4616704           write_min_max: 1 8388608
IPC requests: 0 0                 IPC async requests: 0 0
IPC replies: 0 0                  Open files: 0
```

Figure 5. Example of /proc/fs/dvs/mounts/[0-x]/stats

```
uid: 18639, apid: 450546, jobid: 14941.grace01.nersc.gov, cmdname: /bin/hostname dvs
dvs['/global/scratch2', ' 4172 0', ' 149 0', ' 149 0', ' 0 0', ' 149 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 746 0', ' 0 0', ' 892 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 8 3360', ' 0 0', ' 0 0', ' 0 0', ' 0']['/project', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 1 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0']['/global/u2', ' 1376 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0
0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 1 0', ' 0 0', ' 0 0', ' 0 0', ' 0
0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0']['/global/u1', ' 14343 0', ' 552 0', ' 552 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0
0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 553 0', ' 1104 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0
0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 177', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0']['/global/common', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 1 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', '
0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0']['/dsl', ' 34035 0', ' 47428 0', ' 47329 0', ' 628 0', ' 0 0', ' 0 0', ' 0 0', ' 0
0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 772 0', ' 0 0', ' 0 0', ' 483 0', ' 0 0', ' 0
0', ' 0 0', ' 16440 0', ' 16440 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 0 0', ' 8 32768', ' 0 0',
' 0 0', ' 0 0', ' 0 0', ' 99'
```

Figure 6. RUR output from the dvs plugin