Detecting System Problems With Application Exit Codes

Nicholas P. Cardo
Lawrence Berkeley National Laboratory
National Energy Research Scientific Computing Center
cardo@nersc.gov

ABSTRACT: *With today's large systems, it is often difficult to detect system problems until a user reports an unexpected event. By analysing application exit codes and batch job stderr/stdout files during batch job exit processing, it possible to detect and track system related problems. A methodology was developed at the National Energy Research Scientific Computing Center and implemented through custom utilities to detect and track system problems. The details of this methodology along with the tools used will be discussed in detail in this paper.*

KEYWORDS: XT4, System Errors, Application Failures

1 Introduction

User applications can fail for a large variety of reasons including user error and system problems. As systems grow in scale, the ability to quickly identify system related or node health problems diminishes. Problems can manifest themselves in many ways from the obvious where a node or the entire system fails, or more subtle ways by simply causing applications to exit.

At NERSC, an effort has been put forth to attempt to identify why an application failed. This effort must keep in mind that some applications will fail, but what is to be considered to be normal needs to be identified.

2 Data Collection

The first question to answer was whether or not an application failure can be detected. Once a solution is found then the second part of identify why it failed becomes the challenge.

2.1 Application Exit Codes

It turns out that application exit codes are passed back to `aprun,` whose exit code reflects the application exit status. Therefore, if `aprun` exits with a non-zero exit code, then there is a high likelihood, something failed while running the application. It is possible for an application to exit normally with a non-zero exit code, which would cause a false hit for a failed application. However, this is not a common occurrence.

Furthermore, applications are normally submitted through a workload manager, further complicating matters. It is possible for the batch processing script to encounter errors resulting in no application being launched.

Under XT 2.0, process accounting is based on the BSD version 3 specifications. As such it includes many fields that can help in identifying a failure and tracing that failure. The exit code of the process is retained in that accounting structure, so identifying `aprun` processes that have non-zero exit codes is the first start. This is complicated because `aprun` launches a second `aprun` shepherd process. The exit code from the primary `aprun` command is needed.

2.2 Process Accounting

Linking an `aprun` process with a specific batch job is challenging, but not impossible. The process accounting structure includes parent process ids, making it possible to recreate a process tree where the root is the start of the batch job.

The structure for BSD version 3 accounting records is:

```
char  ac_flag;      /* Flags                */
char  ac_version;  /* ACCT_VERSION         */
__u16 ac_tty;       /* Control Terminal     */
__u32 ac_exitcode; /* Exitcode             */
__u32 ac_uid;       /* Real User ID         */
__u32 ac_gid;       /* Real Group ID        */
__u32 ac_pid;       /* Process ID           */
__u32 ac_ppid;      /* Parent Process ID  */
__u32 ac_btime;     /* Creation Time        */
#ifdef __KERNEL__
__u32 ac_etime;     /* Elapsed Time         */
#else
float  ac_etime;    /* Elapsed Time         */
#endif
comp_t ac_utime;    /* User Time            */
comp_t ac_stime;    /* System Time          */
comp_t ac_mem;      /* Avg Memory Usage     */
comp_t ac_io;       /* Chars Transferred   */
comp_t ac_rw;       /* Blocks Read/Write   */
comp_t ac_minflt;  /* Minor Pagefaults     */
comp_t ac_majflt;  /* Major Pagefaults     */
comp_t ac_swaps;    /* Number of Swaps      */
char  ac_comm[ACCT_COMM]; /* Command      */
```

The key fields that are needed are `ac_pid`, `ac_ppid`, and `ac_comm`. From these it is possible to reconstruct a process tree.

The TORQUE `epilog` is passed the session identifier as the fourth argument to the `epilog`. The session identifier is the process identifier of the process group leader for that batch job. This means that a process tree can be linked back through parent process ids until `ac_pid` is equal to the Session ID.

The result is that an `aprun` in the process accounting file can now be linked back to a batch job.

2.3 *Batch Job Automation*

The key now is to automatically analyse a batch job's exit status as well as the exit status from any application run within it.

With TORQUE as the workload manager, the ability to design an epilogue that will automatically run at the end of any batch is standard functionality.

We now have a means to a means to automatically launch a check of a batch job.

2.4 *apinfo*

The utility, `apinfo`, was developed to walk the process accounting data at the end of every batch job and identify the `aprun` process records for each application launched during that batch job.

This now provides the means to determine if further analysis of the batch job is required. An exit code of 0 for the `aprun` process indicates no failures and thus no further action required.

3 Batch Job Analysis

Now that an application failure could be detected, the second phase is to attempt to automatically identify why.

3.1 *Error Analysis*

Through the course of development, 13 unique conditions were identified.

1. SUCCESS: All `apruns` within a single batch job completed with an exit code of 0. No further analysis required.
2. WALLTIME: The batch job exceeded its requested wallclock time limit.
3. WIDTH: The width parameter for `aprun` exceeds the mppwidth request.
4. NODEFAIL: The application aborted due to a node failure.
5. UNEXBUFFER: The application requires a larger MPICH_UNEXBUFFERSIZE.
6. ENOENT: The `aprun` command could not locate the application to launch.
7. LIBSMA: Shared memory library error.
8. SIGTERM: The batch job was killed.
9. NOTRACE: The processing of accounting data could not match an `aprun` command to the batch job.
10. UNKNOWN: None of the other conditions could be identified.
11. NOAPRUN: The batch did not execute `aprun`.
12. ATOMIC: For a brief time, shmem atomic operations were disabled. This identified applications that killed due to the attempted use of shmem atomic operations.
13. QUOTA: The user exceeded their disk quota.

The information collected is then written out to a daily log file and reports generated.

3.2 *Error Messages*

Common errors can be found in the stdout/stderr files from batch jobs. Simple searches for strings can identify these errors.

- WALLTIME: `"PBS: job killed: walltime"`
- WIDTH: `"exceeds confirmed width"`
- NODEFAIL: `"Received node failed or halted event"`

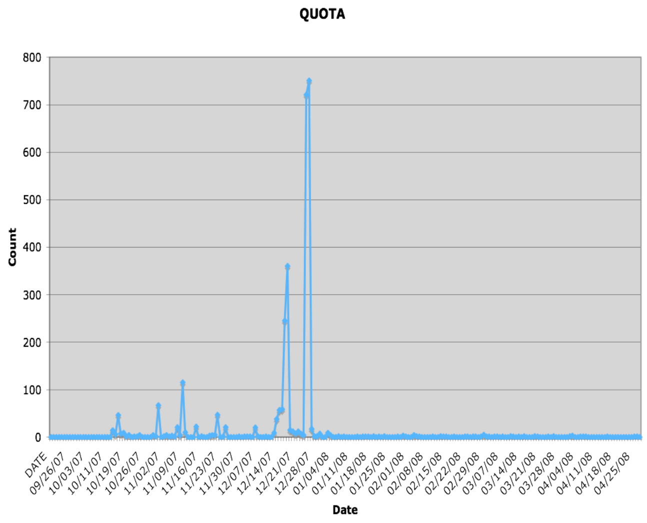
- UNEXBUFFER: `"MPIDI_PortalsU_Request_PUPE(605):"`
- ENOENT: `"No such file or directory"` and `"aprun: file * not found"`
- LIBSMA: `"LIBSMA ERROR:"`
- SIGTERM: `"aprun: Sending caught Terminated signal to application"`

4 Analysis of Job Failures

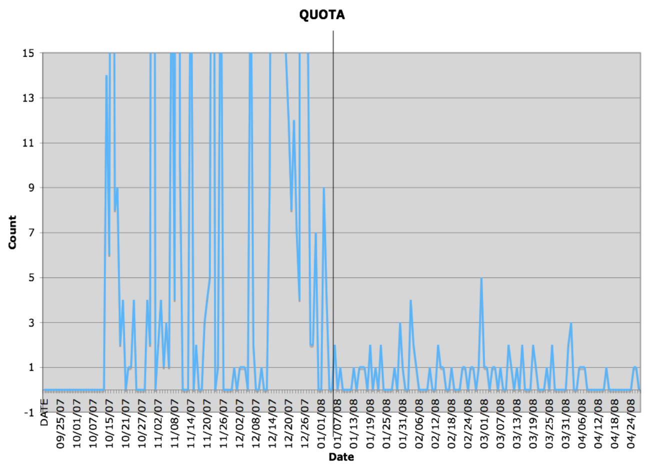
Having the data is the first part to the whole picture. Now it is time to analyze the data to determine if system problems are evident. One thing to keep in mind is that conditions must be applied when analysing the data. In some cases, the actual count of failures is relevant. However, in other cases, looking at the percentage of jobs in that category can show trends as well as what is to be expected.

4.1 QUOTA

Looking at the actual count of jobs that failed due to disk quota exceeded shows the following:



A quick look shows a couple of spikes and what appears to be the expected failure rate. However, taking a closer look shows:

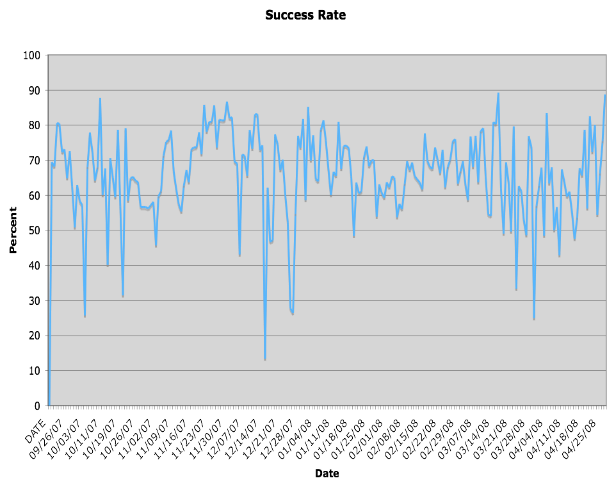


The vertical line represents January 5, 2008. On that date all users we given quota limits of 0, meaning unlimited. All failures after that date could no longer be classified as user error and actually represented a defect in the quota software.

All the high counts prior to that date represent users hitting yet another quota defect, which resulted in inode quotas being enforced at the iunit level, not the actual quota limit.

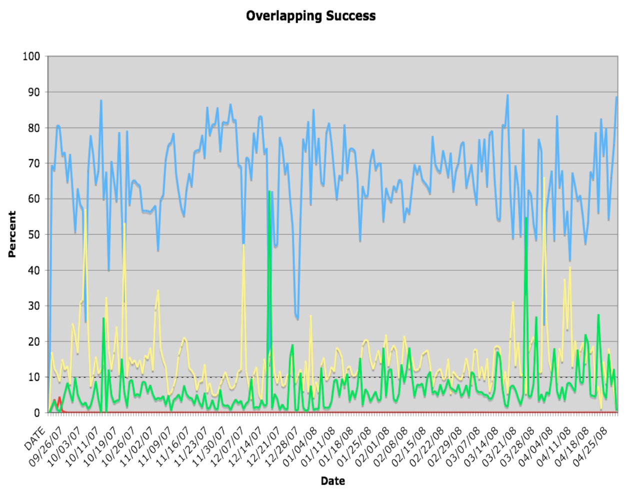
4.2 SUCCESS Rate

By looking at the success rate of jobs, it might be possible to determine the expected success rate as well as any day that requires further investigation.



From this display of data, it is unclear what the expected job completion rate should be. As a starting point, assume the success rate should be at least 70% and now there are days to look at.

By plotting failure rates on the same chart as the SUCCESS rate, we begin to see a pattern.

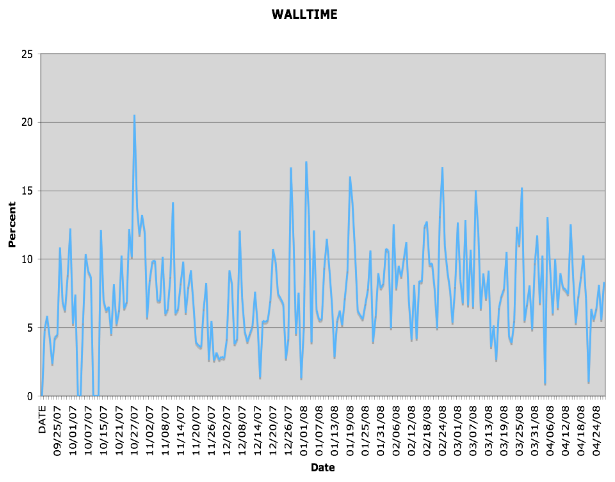


In this case, we are seeing the effects of system wide failures on job completions. Failure rates increase as success rates decrease.

4.3 User versus System Error

Having investigated the various identifiable failed applications, assumptions can now be made to assign the cause of the application failure as either User error or System error. In some cases, the error could represent both.

For example, examining the plot for WALLTIME percentages shows periodic spikes above 10%.

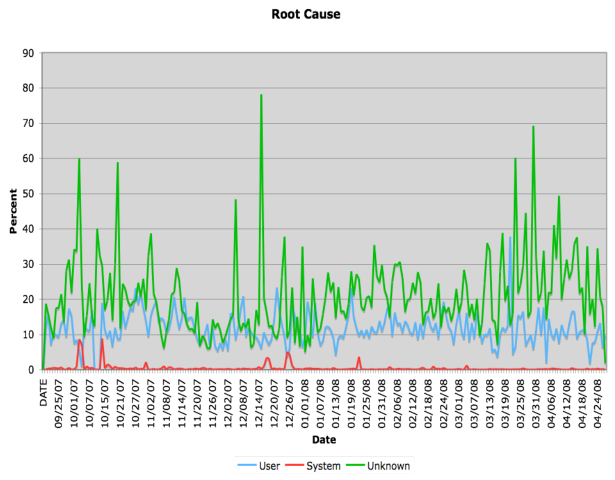


These spikes correspond with user reports of jobs that started but did not make any progress. The normal range appears to be between 5% and 10%. This category contains both user and system root causes. However, there is no way to be 100% certain of the root cause without consulting with the user for each job. This is not practical so trends must be used to identify anomalies.

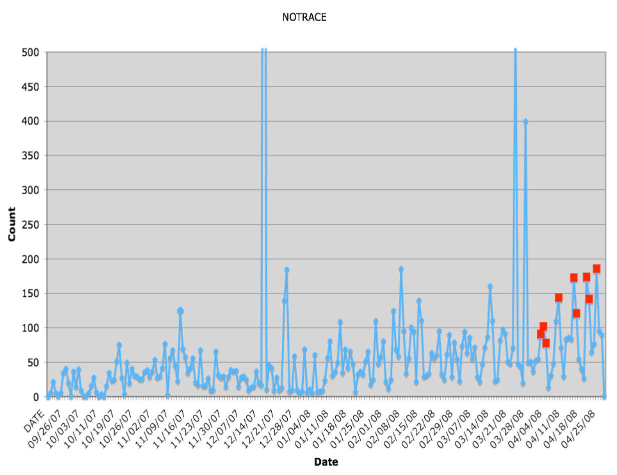
Looking at all the categories, these breakdown into the following explanations.

1. WALLTIME: User and System error.
2. WIDTH: User error.
3. NODEFAIL: System error.
4. UNEXBUFFER: User error.
5. ENOENT: User error.
6. LIBSMA: System error.
7. SIGTERM: Possible system.
8. NOTRACE: unknown root cause.
9. UNKNOWN: Unknown root cause.
10. NOAPRUN: User error.
11. ATOMIC: System error.
12. QUOTA: Currently system error.

Plotting the percent of jobs based on root cause shows the following:



A careful examination of the plot shows minimal system impact. However, careful analysis of the data shows a direct correlation between system failures and jobs classified as NOTRACE. Plotting just the NOTRACEs with indications of system failures in the month of April reveals the following:



The effects of system failures can now be quantified over time through trend analysis.

4.4 UNKNOWN Category

The UNKNOWN category contains all jobs where an error could not be identified. What makes this difficult is the ability of users to redirect STDOUT and STDERR, thereby redirecting any error messages that could be used to automatically determine the error.

5 Daily Reporting

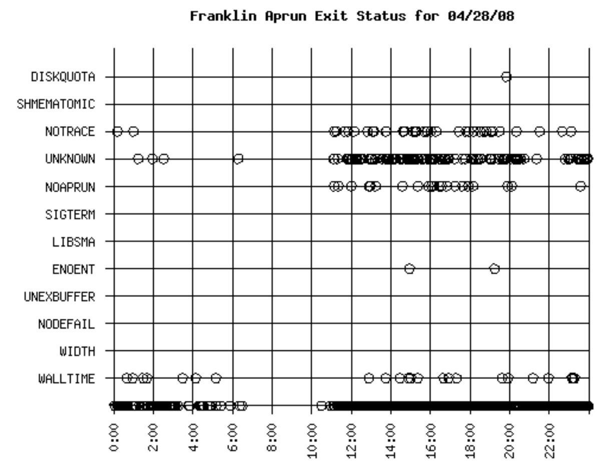
By collecting the data and analysing on a daily basis, it is possible to produce a summary report represent the state of the system for that day. A sample report for NERSC looks like:

```
From: 04/27/08 00:03:14
to:   04/27/08 23:57:58

--------------------------  -----
Exit Status                 Count
--------------------------  -----
APINFO_SUCCESS                676
APINFO_TORQUEWALLTIME          41
APINFO_APRUNWIDTH               0
APINFO_NODEFAIL                 1
APINFO_MPICHUNEXBUFFERSIZE      0
APINFO_ENOENT                   0
APINFO_LIBSMA                   0
APINFO_SIGTERM                  0
APINFO_NOAPRUN                  9
APINFO_UNKNOWN                 28
APINFO_NOTRACE                 18
APINFO_SHMEMATOMIC              0
APINFO_DISKQUOTA                0


Top Ten Failed Users Report
---------------------------
      Count  Username
      -----  --------
          8  user1
          7  user2
          6  user3
          5  user4
          5  user5
          5  user6
          5  user7
          5  user8
          4  user9


Top user in each failed category
------------------------------------------
Exit Code                    CNT  Username
--------------------------  ----  --------
APINFO_TORQUEWALLTIME          2  usera
APINFO_ENOENT                  1  userb
APINFO_NOAPRUN                 7  userc
APINFO_UNKNOWN                14  userd
APINFO_NOTRACE                 8  usere
APINFO_DISKQUOTA               1  userf
```



As normal rates are identified, thresholds can be added to the report to automatically flag situations requiring investigations.

6 Summary

Many errors can be identified and a root cause automatically determined. However, there are a large percentage of jobs that cannot be automatically identified to root cause for failure. The exit processing from `aprun` could be improved to provide a better understanding of jobs failures. When an application exits, the status of that application should be accurately reflected in the exit processing of the `aprun` command. This will allow the reduction in UNKNOWN application failures and provide a more accurate account of job success rates.

It has been shown that by understanding application failures, system errors can be identified. The clearest of this is the Disk Quota Exceeded (QUOTA) category.

Since all users have no limits, there is obviously a problem. Having identified the problem, and SPR can be filed with Cray for corrective action.

Trending analysis can be used to determine the steady state of the system. Deviations from normal trends could indicate a problem in the system. Daily reporting could be used to indicate if the system needs to be examined in detail to see if nodes or subsystems are misbehaving.

7 Acknowledgments

This work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC02-05CH11231.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy.

This work is an adaptation of the methodology developed at Sandia National Laboratory.

8 About the Author

Nicholas P. Cardo is the Project Lead and Lead System Administrator of Franklin. He is a senior member of the Computational Systems Group at NERSC. He can be reached at Lawrence Berkeley National Laboratory, National Energy Research Scientific Computing Center, 1 Cyclotron Rd, bldg 943r0256, Berkeley, CA 94720 USA, E-mail: cardo@nersc.gov.