Reliable Computation Using Unreliable Components

Cray User Group 2013
May 6-9, 2013
SAND2013-3311C

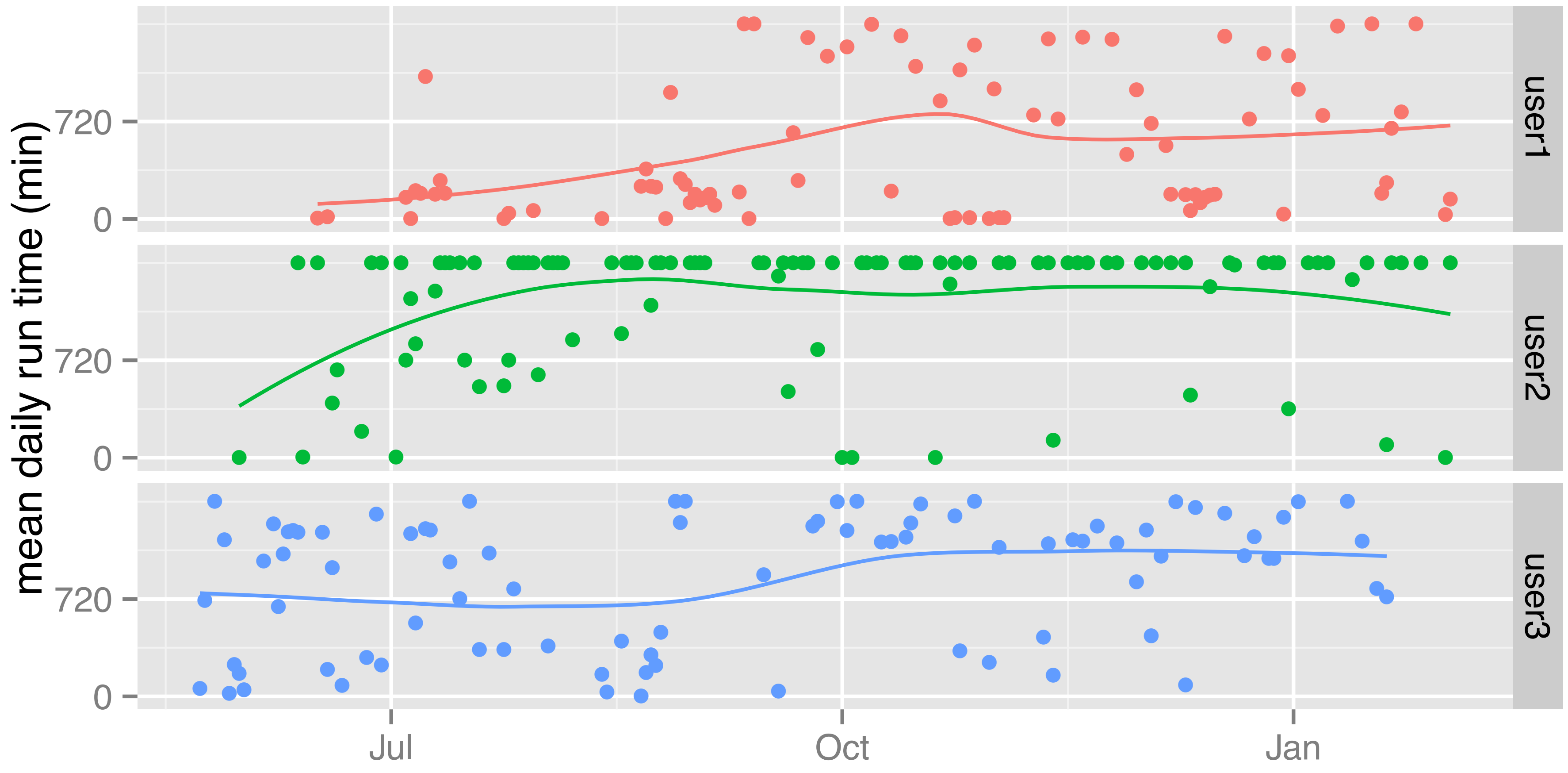
Joel O. Stevenson, Robert A. Ballance, John P. Noe, Suzanne M. Kelly, Jon R. Stearley
Sandia National Laboratories

Michael E. Davis
Cray Inc.

With thanks to Anthony Agelastos, Sandia National Laboratories

Sandia National Laboratories is a multi program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

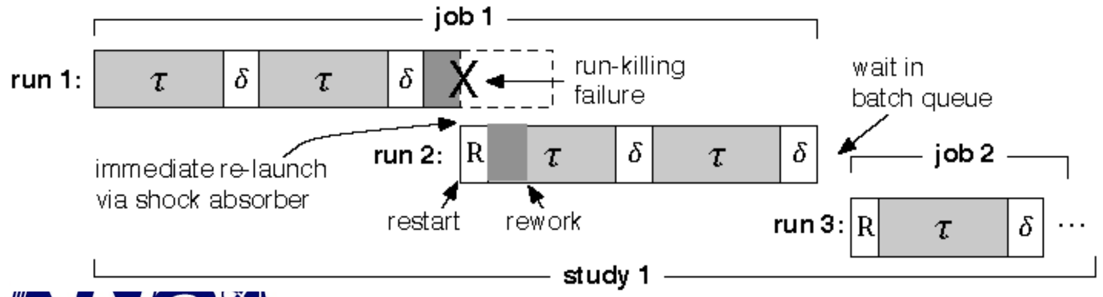
Which User Is Using a Shock Absorber?



Top 3 users over 6 month interval on Cielo

Perspective

- Researchers, engineers, and system administrators are working actively to ensure successive generations of large-scale HPC machines are increasingly resilient to failures
- Adopting the terminology from Snir et al, we define resilience as "*The collection of techniques for keeping applications running to a correct solution in a timely and efficient manner despite underlying system faults*"
- This presentation focuses on user-level resilience techniques that build atop system and application level resilience mechanisms
- Since jobs have a propensity to fail, we need a method to shield a single job execution from perturbing events. The "shock absorber" can be crafted to handle both failures arising from the run (e.g. aprun) and from the environment (e.g. moab).

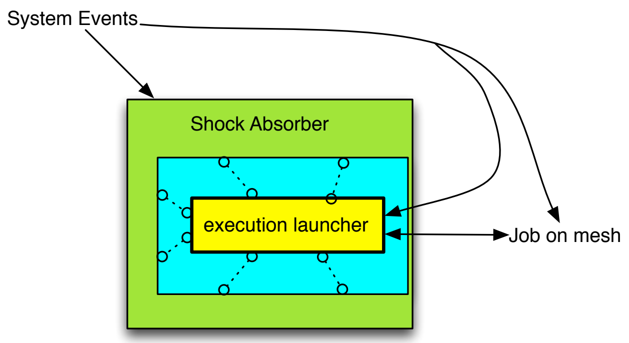


Resilience and Error

- In the real world, events that kill a job can arise from many sources: hardware, software, system reboots, file system glitches, and even user error
- There are four different sources of job execution- killing errors: two in the engineering domain, and two in the human domain
  - p_j is the probability of an isolated job failure
  - p_r is the probability of system reboot when your job is running
  - p_{user} is the probability of a user error
  - p_{admin} is the probability of system administrator error
- Assuming that your study of M batch submissions is always running, the probability of failure-free workload during M job runs is given by
  - $P_{success} = ((1 - p_j)(1 - p_r)(1 - p_{user})(1 - p_{admin}))^M$
  - As M increases, this becomes a (vanishingly) small number

The Shock Absorber Pattern

- **Better designs, better software, better hardware will likely reduce many of these probabilities, but we are not there yet.**
- **The shock absorber evolved to add resilience, at the site, admin, and user level**
- **The basic idea in the shock absorber pattern is to wrap the execution launch command inside a script that can handle system signals execution errors**



```
set_signal_handlers();
outcome = UNKNOWN;
while (nodes_available(...) && time_left(...) &&  outcome == UNKNOWN) {
    try (run_job(...)) {
        catch (all_done()) { outcome = SUCCESS; break; }
        catch (fatal_error()) { outcome = FATAL; break; }
        catch (restartable_error()) {
            // set up for restart & continue loop
        }
}
switch (outcome) {
    SUCCESS: declare_victory(); break;
    FATAL: bad_things_happened(); break;
    GOT_SIGNAL:  signal_happened(); break;
    UNKNOWN:  ran_out_of_nodes(); break;
}
```

Job Submission with the Shock Absorber

- **Jobs submitted using the shock absorber typically reserve several extra nodes so that if a node falls out, the job can relaunch without going back through the scheduler cycle**
  - **Sizing the number of extra nodes is by trial and error, however, if every job reserves extra nodes, the overhead can add up**
  - **A feature that would be highly useful in the batch system would be a call, executed from a service node, to request that additional nodes be added to the current partition. The shock absorber could request new nodes as needed, rather than reserving them ahead of time. One could even envision ALPS doing this for us — when a node is failed in a partition, a new one could be added!**
- **The shock absorber needs to maintain state concerning the number of extra nodes reserved, and the number used, so that it can detect when it runs out of nodes**

What Errors Does the Shock Absorber Handle?

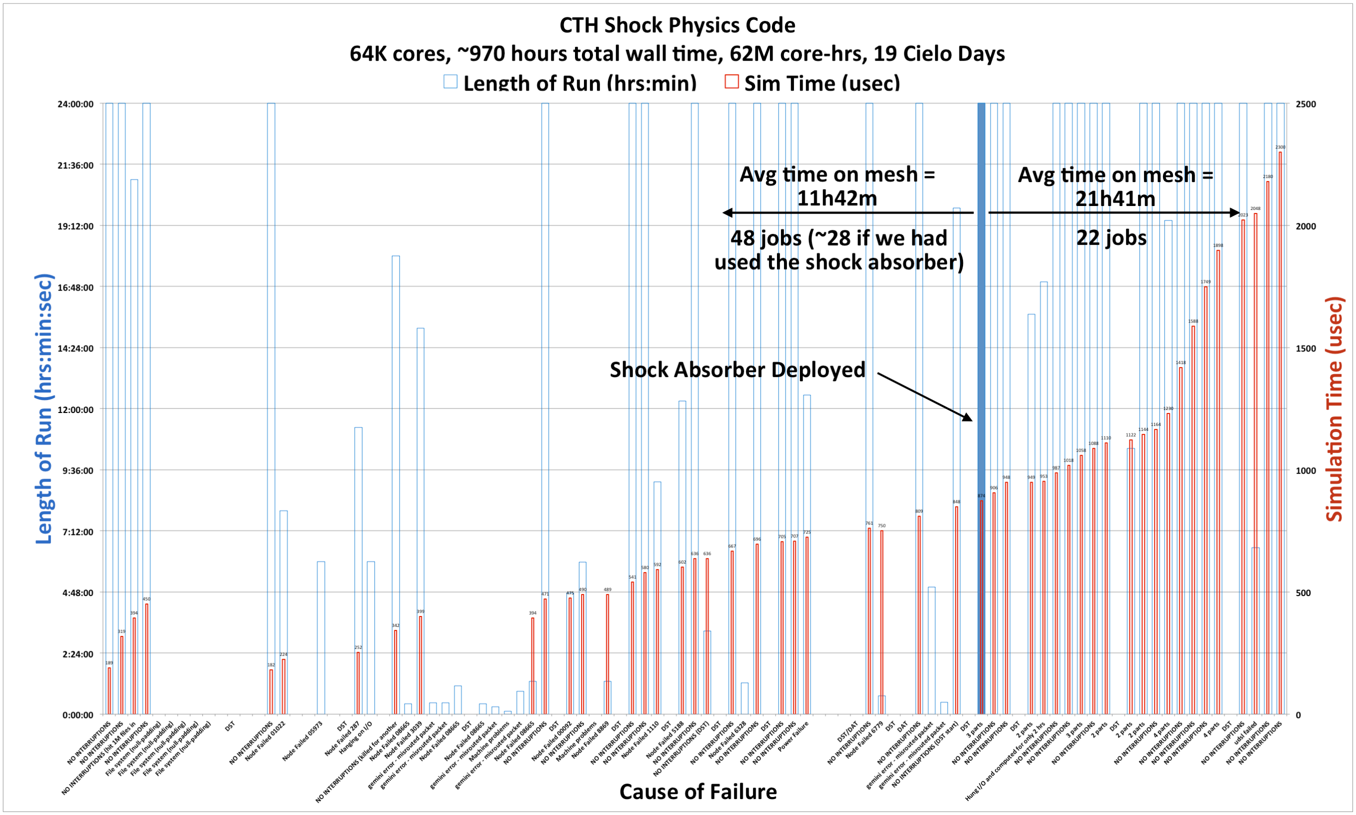
- On Cielo, the generic shock absorber (written as a bash script) handles three key cases
  - ec_node_failed
  - ec_node_halted
  - nem_gni_error_handler
    - If not on a failed node, we manually tie up the node (to take it out of the partition)
    - This error can be detected when the failed node has reported a gemini error; by trial and error we have found that it is best to avoid relaunch on that node

Resilient aprun

- CLE 4.1 introduced ”resilience features” to the aprun command. The features allow aprun, when called with the ‘–R’ option, to retry a job if the job receives either a node failed or node halted event.
- The resilience aspects of aprun are an important step, but the initial deployment has two limitations that are worrisome
  - First, the two errors to be handled seem to be built in. The rules for relaunch would ideally be configurable in a later version.
  - Second, the initial deployment of resilient aprun does not provide (internally) for any job cleanup following a failure
  - There are also potential interaction with the node health checker.

Experience with the Shock Absorber

- **It took 70 runs to achieve 970 hours wall-time (14h/run)**
- **In a perfect world it would take 40 runs (24h/run)**
- **40% ran without error in 24 hours; 43% preventable or recoverable error; 17% unrecoverable**
- **Shock absorber improves app availability and work throughput (11h42m avg run time before; 21h41m after)**



Higher blue bars are better (more time on the mesh)

Other Interesting Topics in the Full Paper

- History of shock absorber pattern
  - SNL has been successfully using this pattern since the early days of Red Storm (2005). The code evolved into ryod.
- When potential job hangs are detected, other tools can be used to capture the job state before a kill signal is sent
  - On Red Storm, the tool was called fast_where
  - In CLE systems, the comparable tool is STAT (Stack Trace Analysis Tool)
  - CLE systems also provide the ATP (Abnormal Termination Processing) tool
- Progress & Rework
  - Workload Progress, Rework, Signals and Grace

Recommendations for Users

- Plan for failures right from the start. Use shock absorbers! They work.
- Keep meticulous records. Knowing the errors that affect your application, their timing and their frequency, will help you to adjust your shock absorber.
- Insist that your application team provide you with good diagnostics. Without clear messages, including well-documented exit codes, you can't improve your response.
- Enjoy life when the system handles the failures for you. As resilience techniques improve, your real" job should get easier.

Recommendations for System Admins

- Implement grace signaling. It takes extra effort to send a grace signal to each job, especially when the grace period can be job-specific. However, this is a matter of scripting.
- Find the root cause for all errors, and maintain a library of recoverable errors with their app-level signature.
- Be wary of system changes that affect job submission scripts. Users employ complicated scripts, so changes to the user environment may have deep ramifications.

Recommendations for System Builders

- Ensure that improvements like aprun -R are extensible with local knowledge. Adding local knowledge is the essential growth path for making smarter systems.
- Work to reduce component and system-level outages
- Assess logging data and make sure that job-level failures (and their cause) can be easily identified and tracked
  - For example, we'd like to find all the job-killing events, and which job they killed. That data takes multiple joins over the data in the current deployment, but we're pretty sure that there's a place where system software knows both the error, and the job to be killed.
- Keep building bigger, smarter, and more robust systems!

Conclusions

- **Human anticipation and imagination will always outrun the capabilities of our HPC systems. No matter how robust and resilient, errors will happen, and design constraints will be exceeded.**
- **The wise user will adapt and employ techniques like the ones described here to their own application, system, and circumstance.**