Fermilab

Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

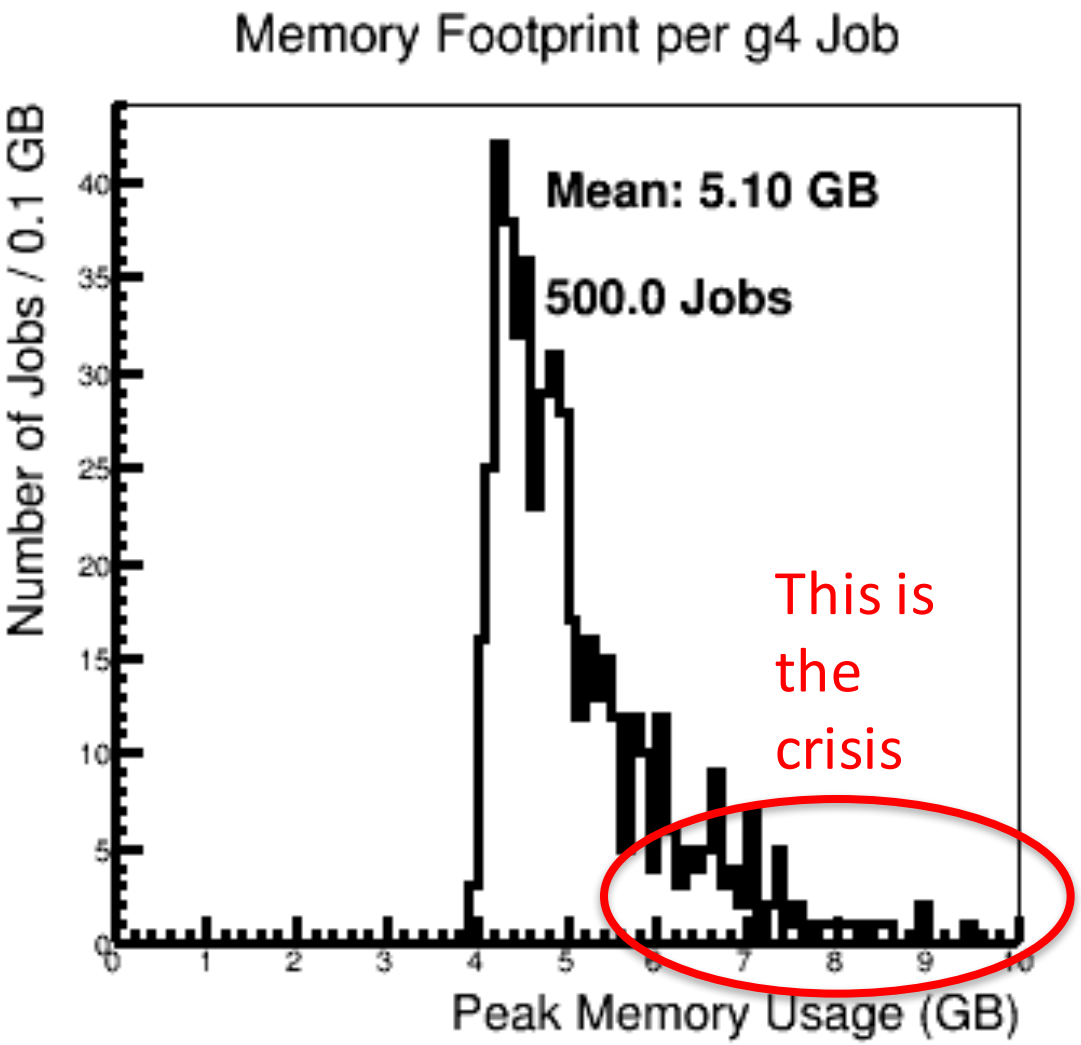
Performance analysis case study: Lessons learned

Chris Jones, Jim Kowalkowski, **Marc Paterno**

HSF Workshop

5 May 2016

Problem



- Memory in the MC workflow jumped significantly.
- First appeared when a new shower generator was used
  - Far more particles in showers

What we did (1)

1. Identify the actual problem
   - Starting from anecdotal reports
   - Blame originally assigned to the wrong target
2. Assemble the team: need the right expertise
   - C++ best practices
   - Codebase being analyzed
   - Performance measurement strategies
   - Use of measurement tools, workflow, managing data
   - Use this as a mentoring opportunity
   - Also needed quick consultations with experiment experts
3. Create validation tests
   - Experiments don't always have enough
   - They caught mistakes!

What we did (2)

4. Make profiling measurements
   - Fast tools are good (internal tools of *art* framework, igprof)
   - For slow tools (valgrind/massif), carefully pick events to analyze
5. Analyze results; identify a few "low-hanging fruits"
6. Make code changes, validate changes using the tests
7. Repeat 4—6 until done
   - How do you know when you're done?
   - We went until there were no more low-hanging fruit
8. Provide modified code to collaboration
   - Easy because of git
   - Easier still using pull requests
9. Put quick memory measurement tool in place as CI test

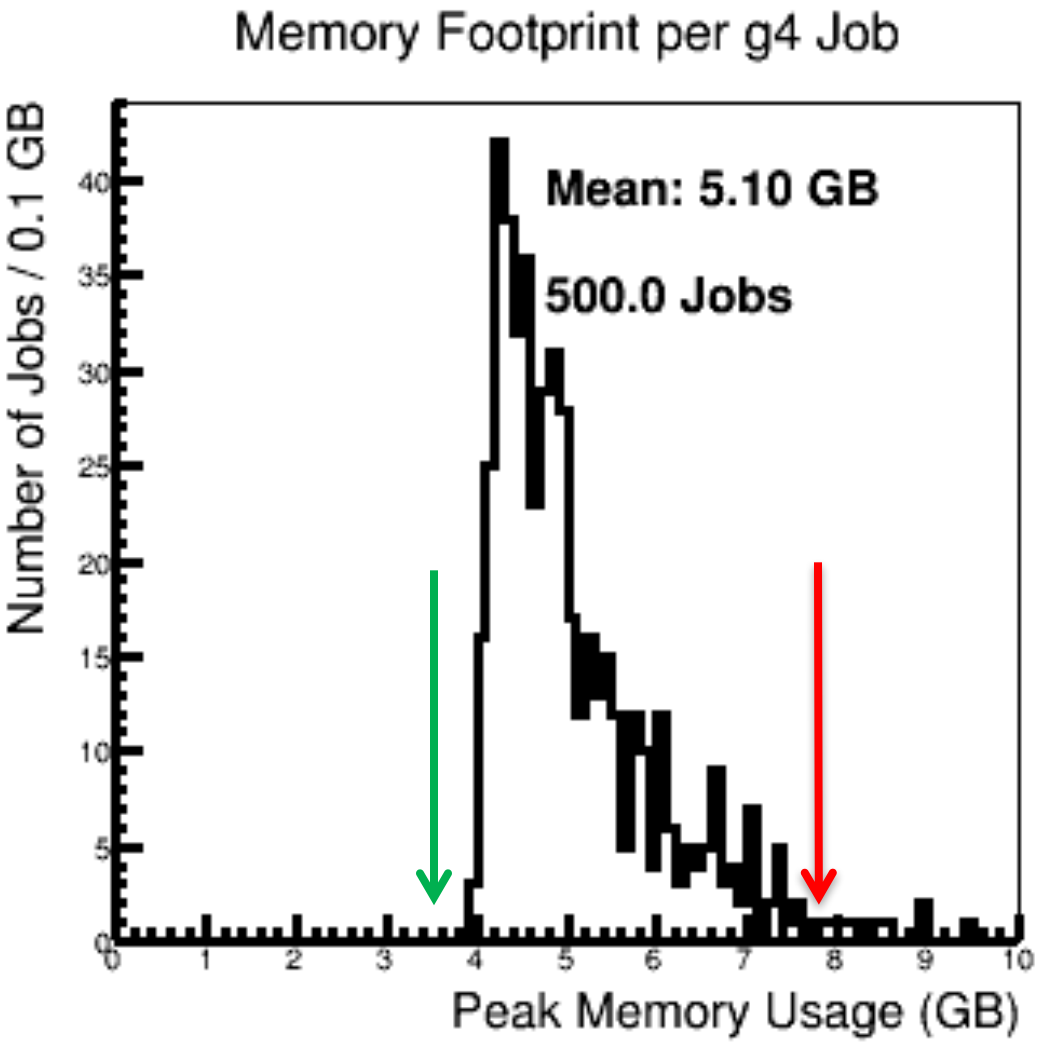
Some code patterns we fixed

- We looked at the *design* and *use* of data structures
  - Some data structures were redesigned to be merged and compacted, e.g.:
    - `vector<vector<float>>` → `vector<float>`; possible because the original inner vector is always the same length,
    - replace "parallel maps" with a single map with a structured `value_type`.
  - Some fixes in basic handling of data structures, e.g.:
    - return a `const& map<…>` rather than `map<…>`.
- We looked at workflow organization issues.
  - Don't cache results in framework module state.
  - Summarize data as early as possible, and keep only the summary, e.g.
    - simulated particles are dropped ASAP when filtering by volume

How we tested

- We were only making changes to organization and handling of data structures.
- We were making no changes to "physics algorithms": no changing thresholds, no changing `double` to `float`, etc.
- We expected exactly the same output from modified code.
- We needed to do enough testing to convince the experimenters to take the code; they would do their own final validation.
- We wrote framework modules to produce ASCII dumps of reconstruction products, and then *diff*ed the resulting files to verify that no changes were introduced.

Summary - performance



- Sample events originally had ~8 GB memory footprint
- After changes, footprint reduced to ~3.6 GB
- Larger scale run needed to reproduce this distribution
- Expected to shift and compress, unclear where that new peak will be.
- Our high-priority effort took about two weeks to complete.

Lessons learned: technical things

- Don't clear memory at the front of calls into member functions that cache data.
  - Clear out the cache as soon as the data is not needed
  - Clearing at the front causes the previous event's data to be retained
  - Recommended practice: don't cache data in algorithmic objects
- Complex data structures used for small look-ups cost memory
  - Maps of numbers to vectors
  - Vectors of maps
  - Vectors of vectors
  - Maps when simple linear structures will do
- All of these hurt when the number of live instances goes way up, and this is what happened with the additional particle flux (latent data structure problems)

Lessions learned: process (1)

- Direct modification of the code was critical for our success.
  - Previous reviews, resulting in a document with recommendations for change, were not as successful.
  - We were able to do this because the experiment needed the fixes, and they needed them quickly.
- Good testing is needed for this kind of work. What constitutes "good" depends on the task, and on who it is done for. Final testing is always in the hands of the experiment.
- CI testing is useful, but not enough.
- We need to catch the design issues earlier to avoid a crisis: better communication, earlier in development.
  - Can we make software development more collaborative, like the development of an analysis?

Lessons learned: process (2)

- Knowledge of fundamental features of data structures is important.
  - At least a cartoon-level understand of overhead from vector, that maps are node-based, etc.
- We need to spread this knowledge.
  - To be added to any future *art* or LArSoft course.
  - Give talks at collaboration meetings.
  - This way of making performance improvements (concrete changes done in a short time) motivates people.

Thanks