Using Performance Tools to Support Experiments in HPC Resilience

Thomas Naughton[1,2], Swen Böhm[1], Christian Engelmann[1] & Geoffroy Vallée[1]

[1] Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN, USA.
{naughtont,bohms,engelmannc,valleegr}@ornl.gov

[2] School of Systems Engineering
The University of Reading, Reading, UK.

Motivation

- Motivation for this work
  - Research on HPC Resilience for large-scale systems
  - Tools to help developers of Fault Tolerance & Resilience

- Leverage “performance tools” to aid resilience community
  - Gather execution data
  - Diagnose problems

Performance Tools

- Performance tools are common in HPC environments
  - Trace tools, Profilers, etc.
  - Care taken for efficiency/scalability
  - Integrate with standard tool-chain
- Benefits in context of resilience
  - Performance + Resilience (costs of FT/R)
  - Global Context for fault injection experiments
- Challenges and considerations
  - Control experiments to ensure data written to stable storage
  - Example: Failed MPI process ➔ ensure trace buffers flushed

Overview

- Background
  - MPI-FTWG / ULFM
  - MPI Trace library
- Experiments
  - ULFM enhanced application
  - ULFM enhanced trace library
  - Trace data usage examples

Background: MPI

- MPI Fault Tolerance
  - MPI Fault Tolerance Working Group (MPI-FTWG)
  - Proposed changes to MPI specification to define necessary set of enhancements for supporting fault-tolerance with MPI
  - "ULFM" – User Level Failure Mitigation proposal
    - Full specification at MPI-FTWG Trac & http://www.fault-tolerance.org
- Failure model
  - Currently limited to MPI process failures (fail-stop / fail-recover)
    - We generally just refer to as "failures".
  - Logical processes as defined by MPI, regardless of implementation

ULFM API: Errors & Setup

- Specification defines two new MPI error classes
  - MPI_ERR_PROC_FAILED
  - MPI_ERR_REVOKED
- For “any source receives”
  - Potential sender failures raise MPI_ERR_PENDING
- Use existing MPI error handlers to take advantage of ULFM
  - By default, set to MPI_ERRORS_ARE_FATAL
  - Must change communicator’s handler to MPI_ERRORS_RETURN
  - Will be notified upon process failures & can address appropriately

ULFM API: Failure Acknowledgement

- Method to “recognize” failures as known & quell future notifications that would occur with MPI_ANY_SOURCE receives
- Also can query for previously “recognized” failures
- Entirely local operation

MPI_COMM_FAILURE_ACK(comm)

IN	comm	communicator (handle)

MPI_COMM_FAILURE_GET_ACKED(comm, failedgrp)

IN	comm	communicator (handle)
OUT	failedgrp	group of failed process (handle)

ULFM API: Communicator Shrink

- Method create a new communicator with failed ranks removed
- Analogous to MPI_Comm_split() with “live” ranks in newcomm
- Collective operation that removes any failed ranks during call, and returns **consistent** “new” communicator at all sites
- *Must call MPI_Comm_revoke() first**
  - *Revoke requirement may be removed in final ULFM spec*

MPI_COMM_SHRINK(comm, newcomm)

IN	comm	communicator (handle)
OUT	newcomm	communicator (handle)

ULFM API: Communicator Revocation

- Method for revoking a communicator & propagate failure notices
- Deadlock avoidance to interrupt collectives
- Must call MPI_Comm_shrink() to obtain usable communicator

MPI_COMM_REVOKE(comm)

IN	comm	communicator (handle)

NOTE: Not required to call revoke upon process failure if not need collective communication (i.e., pt-2-pt still usable until revoke communicator).

ULFM API: Agreement

- Method for consensus about a given value
- Collective call with “logical and” for in/out flag

MPI_COMM_AGREE(comm, flag)

IN	comm	communicator (handle)
INOUT	flag	boolean flag

MPI_COMM_IAGREE(comm, flag, req)

IN	comm	communicator (handle)
INOUT	flag	boolean flag
OUT	req	request (handle)

ULFM API: Quick Summary

- Use MPI error handlers
  - MPI_ERRORS_RETURN
- MPI Error classes
  - MPI_ERR_PROC_FAILED
  - MPI_ERR_REVOKED
  - MPI_ERR_PENDING
    - (for “any source”)
- New MPI functions
  - MPI_COMM_FAILURE_ACK
  - MPI_COMM_FAILURE_GET_ACKED
  - MPI_COMM_SHRINK
  - MPI_COMM_REVOKE
  - MPI_COMM_AGREE
  - MPI_COMM_IAGREE

MPI Trace Tool

- *DUMPI*
  - MPI tracing library to record application execution (function calls)
  - Implemented as PMPI interposition library
  - Developed at Sandia as part of SST project

- Overview
  - Supports individual functions, performance counters (e.g. PAPI)
  - Records function input arguments and (some*) return values
  - Traces recorded as binary files w/ utilities to convert to text
  - Tools to convert to OTF* for visualization in tools like Vampir

** Note: Not sure about current status/support for retvals and OTF converter.*

Modification for experiments

- Enhanced DUMPI library to recognize ULFM API
  - Trace library & text converter tools
- Extended demo application to support ULFM
  - "simpleMD" – molecular dynamic application with app checkpt/restart
  1. Change all MPI_COMM_WORLD to "smd_comm" handle
  2. Change error handler to MPI_ERRORS_RETURN
  3. Modify main simulation loop to recognize process failures (MPI_ERR_PROC_FAILED / MPI_ERR_REVOKED)
  4. On error, all call MPI_Comm_revoke() & MPI_Comm_shrink()
  5. Replace "smd_comm" handle with *newcomm* from shrink
  6. Roll back to previous iteration & continue from previous checkpoint

Test Setup

- Testing platform
  - Single desktop
  - Linux cluster
- Software
  - MPI-FTWG's Open-MPI based prototype of ULFM
  - DUMPI (19mar2013) cloned Mercurial repo w/ ULFM support
- Simulated rank failure
  - If environment variable set, a rank will terminate
  - Application calls `exit()` to simulate abnormal MPI failure,

    i.e., not call MPI_Finalize()
  - This also allows DUMPI library to fire exit handler to write data

Test#1 Performance+Resilience

- Gathered “simpleMD” traces on cluster with & without failures
  - Confirmed that no ULFM functions used in non-fail case
  - Noticed duplicate (extra) calls to MPI_Comm_revoke() in failure case
    - Only 1 required but had 2 that were called in app
  - Time for MPI_Comm_shrink() on 32-node cluster was small (< 0.5 sec)
  - But time in two MPI_Comm_revokes (1 plus, extraneous) was 2-5 secs
    - Approx 1-2% of job walltime w/ single induced failure
- This trace data was useful for seeing overhead of resilience mechanism in our application. We can also see that cost for restart is much higher with this application than the cost for ULFM notificaion & communicator recovery

Global Context for Fault Injection

- Trace data provides diagnostic information
  - Check status if application hangs
  - Gain insights into overall job status during FI experiments
    - Beyond just the single target “victim”
    - Example: what other apps were waiting for that victim, will enter ULFM functions (would be good to have return codes for this)
  - Help to understand detection/notification problems

Example #1 – steps for test

- Test of simpleMD+ulfm+libdumpi and induce a failure
- The steps for the test are:
  1. Set an environment variable to signal a failure at runtime.
  2. Start application with 3 ranks
  3. Application runs for 1000 timesteps
  4. At timestep=500, & victim-rank = 1, simulate a failure w/ `exit()`
  5. Ranks 0 and 2 detect process failure via ULFM return codes, call revoke and shrink to get newcomm, re-initialize data & restart from saved checkpoint
  6. Program finishes & see same result for last timestep non-failure case.

Example #1 (cont.)

- After test we noticed something in output & trace files
  - The final result matched non-failure value — Expected
  - Rank 0 – finish with MPI_Finalize() — Expected
  - Rank 1 – (force-failed) had shorter trace — Expected
  - Rank 2 – had truncated trace & no finalize — Un-expected
- We reviewed ULFM changes to application (ok)
- Then noticed error in our fault-inject logic when used with ULFM

Example #1 - corrected steps for test

- Corrected steps shown in blue
- The steps for the test are:
  1. Set an environment variable to signal a failure at runtime.
  2. Start application with 3 ranks
  3. Application runs for 1000 timesteps
  4. At timestep=500, & victim-rank = mcw_size - 1, simulate a failure w/ `exit()`
  5. Ranks 0 and 2 detect process failure via ULFM return codes, call revoke and shrink to get newcomm, re-initialize data & restart from saved checkpoint
  6. Program finishes & see same result for last timestep non-failure case.

Example #2 – detection

- Our initial tests were done on single machine (see example#1)
- When moved to cluster encountered unexpected hangs when forced failure using same steps as before
- After review of trace data, two observations:
  1. All ranks were few steps beyond failed rank (based on trace timesteps) and all live ranks were at the same timestep in blocking collective
  2. We could see that none of the expected ULFM routines for communicator revocation or shrink had occurred.
- The absense of ULFM revokes explained hangs in collective.
- Realized failure notification wasn't be propagated
  - Change: FI from exit() ➔ MPI_Abort(MPI_COMM_SELF, -1)

Comments

- Trace data helped to provide global context
- Postmortem data helped highlight simple error, but not noticeable by output values due to FT mechanisms doing what they should!
- Possibly could use communication patterns from trace info to automate site selection in FI experiments

Related Work

- BIFIT
  - HPC application “soft errors” Fault injection tool based on *PIN* binary rewriting utilty
  - Leveraged memory profiling in performance tools to identify candidate FI sites
- AutomaDeD
  - Combines sampling & classification /clustering methods to identify abnormalities, possible fault signatures (bugs)
  - Aid debugging parallel apps
- SST/Macro
  - Performance simulator
  - Study hardware characteristics on application performance
  - DUMPI
- xSim
  - Performance investigation tool
  - Extended to support resilience experiments (fault inject, ULFM, etc.)

Summary

- Many tools for performance evaluation & experiments in HPC
  - Can be used to help develop “resilience tools”
- Described:
  - MPI-FTWG “ULFM” & DUMPI-ulfm trace library
  - Extended application for resilience tests
- Trace data was useful for postmortem info for resilience tests
- Future: plan to add tracing support to xSim
  - More control over app timing, failure scheduling and I/O buffering.
  - Possibly see if trace files could be useful for “replay” (study interleaving)

Thank you & enjoy the conference

- This work was supported by the U.S. Department of Energy, under Contract DE-AC05- 00OR22725.
- This work was supported by ORNL National Center for Computational Sciences (NCCS).