The Impact of a Fault Tolerant MPI on Scalable Systems Services and Applications

Richard L. Graham, Joshua Hursey, Geoffroy Vallee, Thomas Naughton, Swen Boehm

Oak Ridge National Laboratory
{rlgraham,hurseyjj,valleegr,naughtont,bohms}@ornl.gov

Cray Users Group – April 29-May 3, 2012

Presented by: Joshua Ladd

Outline

- Motivation & Problem Statement
- Algorithm-Based Fault Tolerance Techniques
- Run-Through Stabilization Proposal to the MPI Forum
- Runtime Environment Requirements
- Cross-Cutting Requirements
- Early Experimentation Results
- Conclusion

Fault Tolerance and HPC

- **Fault tolerance is important to HPC applications**
  - Large scale and long runtimes lead to increased opportunity for failure to disrupt the application (MTTI, MTBF, …)
  - Projected that process failure will become a *normal* event in the future
  - C/R techniques alone will not be enough to handle the rate of failure
    - **Natural & Algorithm Based Fault Tolerance (ABFT)**
      e.g., checksums stored in peers, rewinding computation, redundant computation
- **Entire HPC software stack lacks support for portable, fault tolerant applications.**

International Exascale Software Project (IESP):
Fault Tolerant MPI (2012) → Applications (2016)

Dongarra, J., Beckman, P., et al., "*The International Exascale Software Roadmap*," International Journal of High Performance Computer Applications, 2011 (to appear).

Motivation & Problem Statement

- **The entire HPC software stack lacks support for portable, fault tolerant applications.**
- **The MPI Forum is developing interfaces and semantics that will allow an application to manage failures during execution.**
- **However, the requirements of a fault tolerant MPI application will stress the capabilities of existing system services.**
  - These services will need to evolve and develop novel interfaces to work effectively with scalable, fault tolerant applications.

Outline

- Motivation & Problem Statement
- **Algorithm-Based Fault Tolerance Techniques**
- Run-Through Stabilization Proposal to the MPI Forum
- Runtime Environment Requirements
- Cross-Cutting Requirements
- Early Experimentation Results
- Conclusion

Algorithm-Based Fault Tolerance (ABFT) Techniques

- **Faulty Subgroups**
  - Ensemble-style applications
  - Extensive reliance on error handler
- **Recovery Blocks**
  - Iterative applications
  - Execution block followed by an acceptance test
- **Linear Algebra Libraries**
  - Encapsulate fault tolerant versions of commonly used linear algebra operations.
  - FT-LA project to support ScaLAPACK

```
1  int rc, allsucceeded;
2
3  // Recovery Block
4     rc = MPI_Allreduce( ..., comm );
5     if( MPI_ERR_PROC_FAILED == rc ) {
6        goto acceptance_test;
7     }
8     rc = MPI_Allreduce( ..., comm );
9     if( MPI_ERR_PROC_FAILED == rc ) {
10       goto acceptance_test;
11    }
12
13 // Acceptance Test
14 acceptance_test:
15    // Check result of computation
16    // The return code in this example.
17    allsucceeded = (MPI_SUCCESS == rc);
18    // Agree upon acceptance test
19    MPI_Comm_agree(comm, &allsucceeded);
20    // If failed, then the allsucceeded will be 'false'
21    if( !allsucceeded ) {
22       // Start recovery action
23    }
```

Outline

- Motivation & Problem Statement
- Algorithm-Based Fault Tolerance Techniques
- **Run-Through Stabilization Proposal to the MPI Forum**
- Runtime Environment Requirements
- Cross-Cutting Requirements
- Early Experimentation Results
- Conclusion

MPI Forum's Fault Tolerance Working Group

Define a set of semantics and interfaces to enable fault tolerant applications and libraries to be portably constructed on top of MPI.

- **Application involved fault tolerance** (not transparent FT)
- **Starting with fail-stop process failure**
  - A process failure in which the MPI process permanently stops communicating with other MPI processes, and its internal state is lost.
- **Two Complementary Proposals:**
  - **Run-Through Stabilization:** (*Target: MPI-3.0*)
    - Continue running and using MPI even if one or more MPI processes fail
  - **Process Recovery:** (*Target: MPI-3.1*)
    - Replace MPI processes in existing communicators, windows, file handles

MPI Forum Fault Tolerance Working Group:
https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/FaultToleranceWikiPage

User-Level Failure Mitigation (ULFM) Run-Through Stabilization (RTS) Proposal

- **Failures are managed on a per-communicator basis**
  - `MPI_ERR_PROC_FAILED`: operation failed due to process failure
- **Point-to-Point Communication**
  - Communication between active processes is *unaffected* by the failure of a non-participating process.
- **Collective Communication**
  - Fault-aware: Will not hang in the presence of process failure, but may not return the same return code at all processes.
- **Communicator Creation**
  - Behave as other collectives. Therefore, it is possible that some processes see a valid communicator while others do not.
  - `MPI_COMM_SHRINK(comm, &newcomm)`

User-Level Failure Mitigation (ULFM) Run-Through Stabilization (RTS) Proposal

- `MPI_COMM_SHRINK(comm, &newcomm)`
  - A special fault tolerant creation operation that creates a new communicator with just the alive processes of an input communicator.
- `MPI_COMM_REVOKE(comm)`
  - Any one process can revoke the communication context of a communicator at all processes
  - All subsequent, non-local operations on that communicator will return an error `MPI_ERR_REVOKED`
  - Eventually all other processes will see the error, even if they did not call `MPI_COMM_REVOKE()`.
- `MPI_COMM_AGREE (comm, &flag)`
  `MPI_COMM_IAGREE(comm, &flag, &req)`
  - Collective fault tolerant agreement operation that will return uniformly at all processes with the same return code and value for **`flag`**.
  - **`flag`** is a boolean argument, and agreement takes the logical **`AND`** of all input values.

MPI Forum Fault Tolerance Working Group:
https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/FaultToleranceWikiPage

Outline

- Motivation & Problem Statement
- Algorithm-Based Fault Tolerance Techniques
- Run-Through Stabilization Proposal to the MPI Forum
- **Runtime Environment Requirements**
- Cross-Cutting Requirements
- Early Experimentation Results
- Conclusion

Runtime Requirements for FT in HPC

- Support scalable application launch & management
  - MPI and non-MPI applications
- Reliable out-of-band communications
  - Bootstrapping higher-level HPC communication services
- Process failure detection and notification
- Robust process management and clean-up
- Support for process recovery
- Support scalable & resilient communication
  - Example: Binomial graph (BMG) topology

Outline

- Motivation & Problem Statement
- Algorithm-Based Fault Tolerance Techniques
- Run-Through Stabilization Proposal to the MPI Forum
- Runtime Environment Requirements
- **Cross-Cutting Requirements**
- Early Experimentation Results
- Conclusion

Cross-Cutting Requirements: Interconnection Network

- **Must have well defined semantics and interfaces for managing process, node, and switch failure.**
- **Expose as much information about the failure as possible to assist the layer above in managing the problem effectively.**
  - An MPI library might take different actions if it knew that the process failed versus the switch between the processes failed.
- **Must be able to remove a request from the hardware matching queue**
- **Collective operations must be fault-aware. So hardware collectives might need a feedback mechanism to flush the collective operation when an error occurs.**

Cross-Cutting Requirements: Resource Manager

- **Assume: Once a process failure occurs the application should be terminated (so as not to waste resources)**
  - Runtime must have the ability to override this default behavior, and request that the resource manager allow the application to continue operating after process loss.
  - The runtime is then responsible for assisting the MPI library and application in managing process failures, and notifying the resource manager when it is no longer able to do so.
- **Currently, must be resilient to node failure in order to sustain jobs when nodes outside of their allocations fail.**
  - Expose this resilient out-of-band communication to the runtime and MPI libraries so as to reduce system noise and not duplicate effort.
- **It would be useful to 'give back' resources that are no longer needed to the system.**

Cross-Cutting Requirements: Scheduler

- **It would be useful to 'give back' resources that are no longer needed to the system.**
  - When recovering from failure some applications may decide to remove other alive processes to regain the proper proportion of processes.
- **Fault tolerant applications can benefit from a more dynamic scheduling environment.**
  - The ability to dynamically extend their allocation to replace failed resources.
    - Possibly using a shared pool of nodes that are used by small, pre-emptible jobs to aid in a quick turn around time.
- **Expose an interface to query if the scheduler supports various additional capabilities**
  - For example, releasing allocated nodes or requesting replacement nodes

Cross-Cutting Requirements

- Application must be provided sufficient information to make informed decisions, and interfaces to communicate their needs.
- Documentation of any and all exposed interfaces
- Collaboration and coordination of activities between system software layers

Outline

- Motivation & Problem Statement
- Algorithm-Based Fault Tolerance Techniques
- Run-Through Stabilization Proposal to the MPI Forum
- Runtime Environment Requirements
- Cross-Cutting Requirements
- **Early Experimentation Results**
- Conclusion

Experimental environments

- **Cray XK6 at ORNL**
  - **Production platform: *JaguarPF***
    - **Phase-V of *Titan* upgrade (April 2012)**

Resource	Size/type
Cabinets	200
Compute Nodes	18,688
Compute Cores	29,008 AMD Opteron Interlagos
Nodes w/ GPUs	960 NVIDIA
Total Memory	600TB
Interconnect	Gemini
Peak Perf.	3.3 pflops/s

  - **Development & Testing: *Chester***
    - **Single cabinet XK6 with same specs as *JaguarPF***
      - **80 nodes, 1280 total cores, 32GB/node memory, Gemini, etc.**

* Image courtesy of the National Center for Computational Sciences, Oak Ridge National Laboratory.

Early Experimentation Results: ULFM RTS MPI Prototype

- **NetPIPE Latency/Bandwidth**
  - 1% overhead in shared memory latency
  - Negligible impact on shared memory bandwidth
  - Negligible impact on performance over the Gemini interconnect
- **Collectives:**
  - Existing collectives over point-to-point did not need to be modified
  - The collectives only needed to error out when a failure is encountered
- **Agreement:**
  - Log scaling performance results presented at EuroMPI 2011
  - Performance similar to an MPI_Allreduce over the alive processes.

Hursey, J., Naughton, T., Vallee, G., Graham, R., "*A Log-Scaling Fault Tolerant Agreement Algorithm for a Fault Tolerant MPI*," EuroMPI, 2011.

Early Experimentation Results: Runtime Prototype

- Prototype supports different connection topologies
  - Used to connect runtime agents, e.g., tree, mesh, BMG
    - Example: At launch a tree-based "bootstrap" topology is setup
- Binomial graph (BMG) topology to create redundant communication channels
  - Improves FT & scalability properties
  - To setup the BMG, nodes are mapped to the topology and reuse any existing "bootstrap" links
  - Table shows current BMG mapping times

Nodes	Time (sec)
1024	0.19
2048	0.25
4096	0.47
8192	1.21
16384	3.10
32768	12.68
65536	65.36
131071	297.95

Useful Cray Enhancements for FT Runtime

- ALPS
  - Public interfaces for ALPS with full documentation
  - C library interface to ALPS at both service & compute nodes
    - Specifically to the placement data (e.g., placement_list)
  - Set policy for ALPS to avoid killing all tasks in same app
    - Avoid killing all tasks with same appId if one task fails/aborts
  - Extensible ALPS daemon (e.g, apinit)
    - To avoid duplication for base runtime agents
  - Provide public interface to communication system used by ALPS
- Failure data
  - Publish RAS data about failures
    - Example information about network link & node failure history

Conclusions

- Scientific applications are looking to Algorithm-Based Fault Tolerance techniques to more efficiently manage process failure in exascale systems.
- The User-Level Failure Mitigation Run-Time Stabilization proposal allows an application to continue execution even when one or more processes fail during execution.

To withstand the projected failure rates of exascale systems the entire software stack must be resilient and working in concert. This level of integration and cooperation is necessary to provide the application with every opportunity to efficiently manage failures as they emerge during normal execution so they can best harness exascale class machines.