Semantic and State: Fault Tolerant Application Design for a Fault Tolerant MPI

Edgar Gabriel and Graham E. Fagg

George Bosilca, Thara Angskun, Chen Zinzhong, Jelena Pjesivac-Grbovic, and Jack J. Dongarra
<egabriel, fagg, dongarra>@cs.utk.edu
Overview

- Overview and Motivation
- The FT-MPI specification
- Implementation details
- Performance evaluation
- A fault-tolerant Master-Slave framework
- Tools for FT-MPI
- Ongoing work
- Summary
Motivation

- Trends in HPC:
  - High end systems with thousand of processors
  - Grid Computing
- Increased probability of a node failure
  - Most systems nowadays are robust – machines do not crash because of a node failure
- Node and communication failure in distributed environments

- MPI widely accepted in scientific computing

Mismatch between (robust) hardware and (non fault-tolerant) programming paradigm of MPI.
FT-MPI

» Define the behavior of MPI in case an error occurs
» Give the application the possibility to recover from process-failures
» A regular, non fault-tolerant MPI program will run using FT-MPI
» Stick to the MPI-1 and MPI-2 specification as closely as possible (e.g. no additional function calls)

» What FT-MPI does not do:
  » Recover user data (e.g. automatic checkpointing)
  » Provide transparent fault-tolerance
The FT-MPI specification

» General steps when dealing with fault tolerance:
  » Failure detection
  » Notification
  » Recovery procedure

» Questions for the recovery procedure:
  » How to start recovery procedure?
  » What is the status of MPI objects after recovery?
  » What is the status of ongoing messages after recovery?
FT-MPI communicator modes

» **ABORT**: just do as other implementations

» **BLANK**: leave hole

» **SHRINK**: re-order processes to make a contiguous communicator
  » Some ranks change

» **REBUILD**: re-spawn lost processes and add them to MPI_COMM_WORLD
FT-MPI message modes

» **RESET**: ignore and cancel all currently active communications and requests in case an error occurs. User will re-post all operations after recovery.

» **CONTINUE**: all operations which returned MPI_SUCCESS will be finished after recovery.
RESET and CONTINUE

RESET: a message posted in one epoch does not match a receive posted in another epoch

CONTINUE: epoch argument not regarded for message matching
Collective communication modes

» **ATOMIC**: either everybody succeeds or nobody
  » Good for the user, bad for performance

» **NON-ATOMIC**: if an error occurs the outcome of the collective operations is undefined
  » No input buffer is touched, operation can easily be repeated after recovery
  » User can check whether operation has finished properly (e.g. executing `MPI_Barrier` after operations)
  » Usage of `MPI_IN_PLACE` (MPI-2) can introduce problems
Application view

» Line by line checking

/* check return value */
ret = MPI_Send ( buf, count, datatype, tag, dest, comm );
if ( ret == MPI_ERR_OTHER ){
 /* call recovery function */
}

» Usage of error-handlers

/* install recovery handler just once */
MPI_Comm_create_errhandler (my_recover_function, &errh);
MPI_Comm_set_errhandler (MPI_COMM_WORLD, errh);

/* automatic checking. No modification necessary */
MPI_Send (...)
MPI_Scatter (...)

Architecture

Running under a HARNESS Core

High level services

- Name Service
- Notifier service

MPI application
- libftmpi
- Startup plugin
- HARNESS

MPI application
- libftmpi
- Startup plugin
- HARNESS

One startup plug-in per ‘core’
Architecture

High level services

Running outside of a Core

Name Service

Notifier service

One startup daemon per ‘host’
Latency test-suite (large messages)
Latency test-suite (small messages)

![Latency test-suite on boba-cluster](image)

- FT-MPI
- LAM 7.0
- MPICH 2.0.4
- MPICH 1.2.5

Execution time [ms]

Message Length [Bytes]
PSTSWM benchmark

- Dual P4 cluster, 2.4 GHz, GEthernet
- 32 processes on 16 nodes

![Bar chart]

- MPICH 1.2.5
- MPICH 2-0.94
- FT-MPI cyclic
- FT-MPI block
HPL benchmark

- SGI Altix, 32 Intel Itanium 2 processors, 2.1 GHz
A fault-tolerant master-slave framework

» Useful for parameter sweeps
» Basic concept: Master keeps track of the state of each process and which work has been assigned to it
» Works for the REBUILD, SHRINK and BLANK mode

» Minimizes the lost work if an error occurs
» Implementation in C and Fortran available
» Implementation with and without the usage of error-handlers available

» If master dies, the work is restarted from the beginning (REBUILD) or stopped (BLANK/SHRINK)
Master process: transition-state diagram

- AVAILABLE
- WORKING
- RECEIVED
- FINISHED
- SEND_FAILED
- RECV_FAILED
- DEAD

--cont-
- sent
- recv
- done

- ok
- error
- recover

BLANK/SHRINK: mark failed processes as DEAD
REBUILD: mark failed processes as AVAILABLE
Worker process: transition-state diagram

REBUILD: Master died, reset state to AVAILABLE
Harness console

» HARNESS user interfaces
- Manual via command line utilities
  » hrun or ftmpirun
- HARNESS Console
  » Has much of the functionality of the PVM + the addition to control jobs via ‘job-run’ handles
    » When a MPI job is started all processes get a job number. The whole application can be signed or killed via this number with a single command.
  » Can use hostfiles and script command files
Welcome to the Harness/FT-MPI console

con> conf

Found 4 hosts

<table>
<thead>
<tr>
<th>HostID</th>
<th>HOST</th>
<th>PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>torc4.cs.utk.edu</td>
<td>22503</td>
</tr>
<tr>
<td>1</td>
<td>torc1.cs.utk.edu</td>
<td>22503</td>
</tr>
<tr>
<td>2</td>
<td>torc2.cs.utk.edu</td>
<td>22500</td>
</tr>
<tr>
<td>3</td>
<td>torc3.cs.utk.edu</td>
<td>22502</td>
</tr>
</tbody>
</table>

con> ps

<table>
<thead>
<tr>
<th>ProcID</th>
<th>RunID</th>
<th>HostID</th>
<th>Command</th>
<th>Comment</th>
<th>Status</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4096</td>
<td>20482</td>
<td>0</td>
<td>./bmtest</td>
<td>FTMPI:proces</td>
<td>exited(val:0)</td>
<td>5s</td>
</tr>
<tr>
<td>4097</td>
<td>20482</td>
<td>1</td>
<td>./bmtest</td>
<td>FTMPI:proces</td>
<td>exited(val:0)</td>
<td>5s</td>
</tr>
</tbody>
</table>

con>
MPE with FT-MPI

FT-MPI HPL trace via Jumpshot using MPE profiling
KOJAK/PAPI/Expert

Performance Properties

- Total: 0.0
- Execution: 81.4
- Communication: 1.2
- Collective: 0.0
- Late Receiver: 0.0
- IO: 0.0
- Synchronization: 0.0
- DMP: 0.0
- Idle Threads: 0.0

Dynamic Call Tree

- main
  - MPI_Init
  - MPI_Send
  - MPI_Comm_split
  - MPI_Sendrecv
  - MPI_Type_struct
  - MPI_Type_commit
  - MPI_Ssend
  - MPI_Type_free
  - MPI_Comm_free
  - MPI_Finalize

Locations

- Linux Cluster
  - anakin02
  - Process 2
  - anakin04
  - Process 3
  - anakin05
  - Process 0
  - anakin06
  - Process 1
Current status

» Specification available as written document
» FT-MPI v1.0 released for SC 2003
» Current version implements
  » Whole MPI-1.2
  » Some parts of MPI-2
    » Section 4 (Miscellany) complete
    » Section 8 (External interfaces) nearly complete
  » C++ Interface (IU)
» ROMIO being tested
  » Non-ft version
» Ported to both 32 and 64 bit OS
  » AIX, IRIX-6, Tru64, Linux-Alpha, Solaris, Linux, Windows
    » Data conversion: receiver does it right
» Compilers : Intel, gcc, pgi, vendor-compilers
Ongoing work

» FAMP: Fast Asynchronous Multi-Protocol device (tcp, shmemp, myrinet)
» New datatype engine available
» Automatically tuned collective operations
» MPI-2 dynamic process management

» Abstracting an FT-ADI:
  » Event handling
  » Information management
  » Process management
**MPI and fault tolerance (cont.)**

We need to fix it up here...

... and here...

... and here...

You are here

---

User application

Parallel Load Balancer

Parallel Numerical Library

Do I=0, XXX

MPI_Sendrecv ()

...
Suggestion for improved Error-handlers

» Application/libraries can replace an error handler by another error-handler

» Better: add an additional function which would be called in case of an error
  » e.g. like done for attribute caching or
  » e.g. unix `atexit()` function
Summary

» Fault tolerance for MPI applications is an active research target
  » Large number of models and implementations available
  » Design of FT-MPI is in the “spirit” of MPI
    » Does not harm the performance of MPI applications
    » Supports various usage and fault-tolerance scenarios

» More fault-tolerant applications in Jack’s presentation

http://icl.cs.utk.edu/ftmpi
http://icl.cs.utk.edu/harness
harness@cs.utk.edu
Contact information

» Links and contacts

» HARNESS and FT-MPI at UTK/ICL
   http://icl.cs.utk.edu/harness/

» HARNESS at Emory University
   http://www.mathcs.emory.edu/harness/

» HARNESS at ORNL
   http://www.epm.ornl.gov/harness/