Towards Support for Fault Tolerance in the MPI Standard

LEADERSHIP
COMPUTING FACILITY
NATIONAL CENTER FOR COMPUTATIONAL SCIENCES

presented by

Gregory A. Koenig

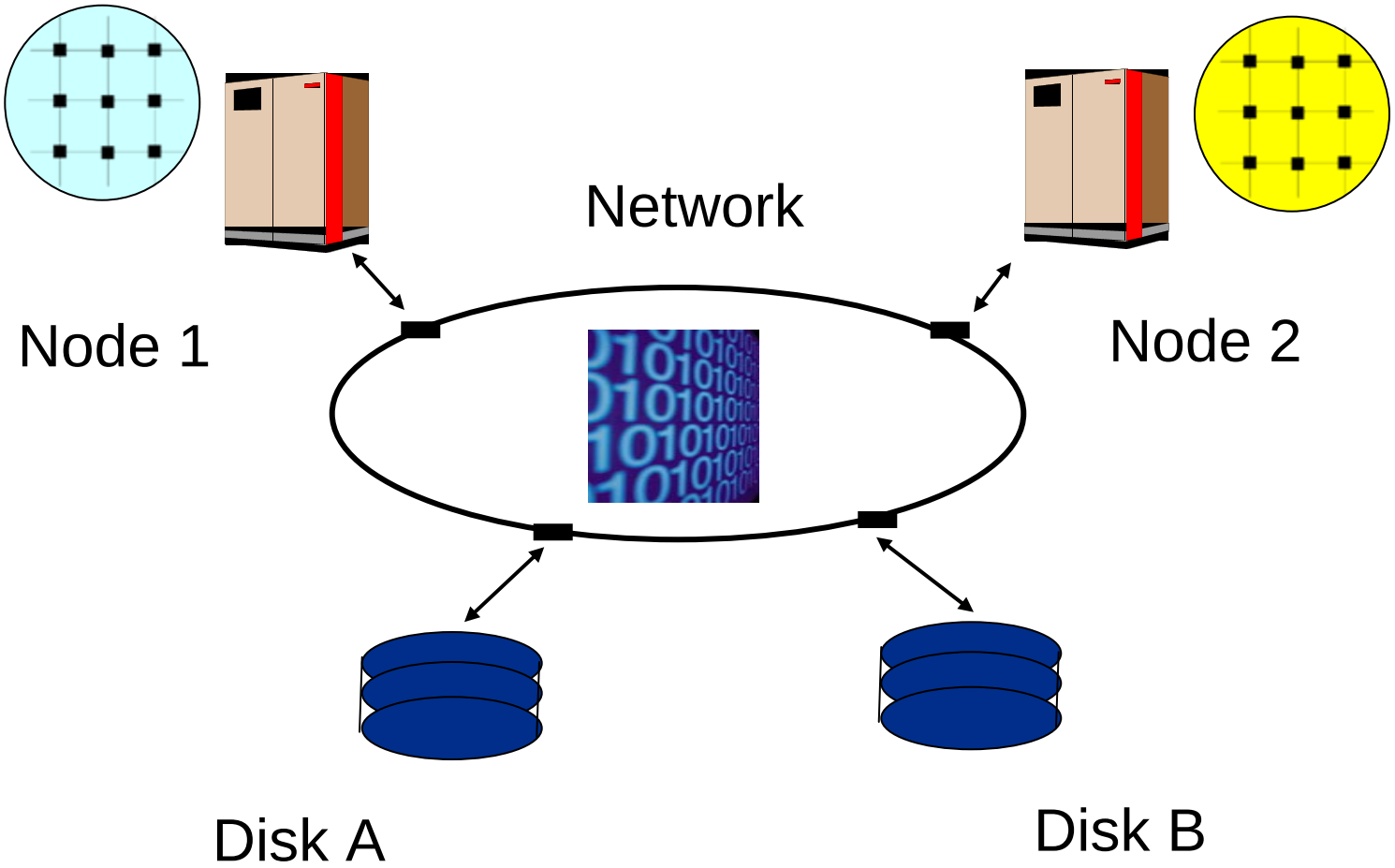
Oak Ridge National Laboratory
U.S. Department of Energy

Problem definition

- **Problem:** The integrity of a component within a running MPI job is compromised due to either a hardware or software fault

- **Question:** Can the MPI application continue to run correctly?
  - Does the job have to abort?
  - If not, can the job continue to communicate?
  - Can there be a change in resources available to the job?

Problem definition



Problem definition … in reality ☺

What role should MPI play in recovery?

- MPI does ***not*** provide application-level fault-tolerance
- MPI should enable the survivability of MPI itself upon failure of a hardware or software component
- MPI provides:
  - Communication primitives
  - Management of groups of processes
  - Access to the file system

What role should MPI play in recovery?

- Therefore, upon failure, MPI should

  (limited by what is practical within the faulted system state):
  - Restore MPI communication infrastructure to a correct and consistent state
  - Restore process groups to a well-defined state
  - Restore process connections to the file system
  - Provide hooks related to MPI communications needed by other protocols that build on top of MPI such as…
    - Flushing the message system
    - Establishing network quiescence
    - Sending “piggyback” data (i.e., annotated user data)
    - Others things?

CURRENT WORKING GROUP STATUS

Error reporting mechanisms (being firmed up now)

- By default, errors are associated with specific MPI requests and are returned synchronously

  Example:

  Ret1 = MPI_Isend (comm=MPI_COMM_WORLD, dest=3, … request=request3)

  Link to 3 fails

  Ret2 = MPI_Isend (comm=MPI_COMM_WORLD, dest=4, …request=request4)

  Ret3 = MPI_Wait (request=request4) // success

  Ret4 = MPI_Wait (request=request3) // error returned in Ret4

  Caller can then ask for more information about the failure

Error reporting mechanisms (being firmed up now)

- Allow the user to register event handlers that are invoked asynchronously for specified…
  - Classes of failures (e.g., link failures)
  - Events associated with particular resources (e.g., failure of a process in a particular communicator)
- Errors are associated with communicators
- Several open questions remain

  (a couple are discussed later in the talk)

Error reporting mechanisms (being firmed up now)

- A collective call has been proposed to check on the fault status of a communicator
  - No extra cost is incurred for consistency checks, unless fault status is requested
  - Provides a global communicator state just before the call is made; if a fault happens in a process immediately after it reports its status as “ok”, this fault will not be identified

Specific use case scenarios

- Process failure in a client/server job

  (client is a member of an *inter-communicator*)
  - Client process fails
  - Server is notified of failure
  - Server disconnects from the client inter-communicator and continues to run
  - Client processes are terminated

Specific use case scenarios

- Process failure in a client/server job

  (client is a member of an *intra-communicator*)
  - Client process fails
  - Processes communicating with the failed process are notified of its failure
  - Application specifies a response to failure
    - Abort
    - Continue with reduced process count, with the missing process being labeled MPI_Proc_null in the communicator
    - Replace the failed process in the communicator
      - Suggestion: it might be useful to *increase* the size of the communicator after a fault

Specific use case scenarios

- Possible steps for continuing with a reduced process count or replaced process
  - Mark affected communicator as being in an error state
  - Discard traffic associated with the failed process
  - “Repair” the communicator
  - Mark the communicator as running
  - Let the application resume
  - Application is responsible for restoring application state
    - Checkpoint/restart
    - Application regenerates state on it’s own
    - ?

Open Questions

- How much heavy-handedness is necessary at the standard specification level to recover from failure in the middle of a collective operation? Is this more than an implementation issue? (Hint: Performance is the fly in the ointment here.)

- What is the impact of “repairing” a communicator on the implementation of collective algorithms? Is it necessary to pay the cost of fault tolerance all the time?

Next Steps

- Complete gathering use case scenarios
- Flesh out the data-piggybacking ideas (needed for some recovery mechanisms)
- Develop first full cut at changes to the standard to support recovering from failed processes
- Prototype and test with applications
- Revisit the proposed standard

For involvement in the process see:

http://meetings.mpi-forum.org/