Support for Developing Fault Tolerant Modular MPI Applications

Greg Bronevetsky

Setting:

The Fault Tolerance API is difficult to use by applications composed of multiple modules. Such applications typically iterate through calls to multiple modules, where different processes may exit a given module at different times even if the code in the module itself performs global synchronizations. As such, it is possible for one process to detect a given failure while inside a call to one module (module F for "Failing") and another process will detect the same failure after it has exited this call and has advanced to calling a completely different module (module R for "running"). Furthermore, it is possible for different modules to use different subsets of processes, which further complicates whole-application error recovery.

Problem 1:

Module R uses a different set of communicators from module F that only partially overlaps with module F's communicators. Thus, we require an out-of-band failure notification mechanism that will inform module R of the failure where its process needs to participate in the recovery of module F, even if module R's communicators do not include the failed process.

Problem 2:

Need a simple way for the application to reach a state where all processes return to module F to perform F's error recovery protocol without forcing other modules, such as R, to be explicitly aware of the fact that the application is using module F or any other particular module.

Solution 1:

An out-of-band error notification mechanism that allows any application process to notify other processes that it needs their participation to perform error recovery. We can use an API similar to active messages where one process can cause a callback function to be invoked on another process.

Solution 2:

An incremental checkpointing technique that allows modules to roll back their progress to the point in time immediately before the call to a given module, including all computational and communication operations.

2.A Incremental Checkpointing

When process p exits from a call to module F, it begins to incrementally checkpoint its state. One implementation of this would be to write-invalidate all application pages when the call to F returns and register a segfault handler. When the application tries to write to an invalidated page, the segfault handler would be invoked and would make a copy of the page before releasing the invalidation and allowing the application to perform its write. When some other process q, which is still inside its call to F, detects that an error occurred during the computation inside F, it will use the out-of-band error notification mechanism to invoke a callback on each process that participated in F's computation. When this callback is invoked on p, it will use its incremental checkpointing log to roll back all state changes

since its last call to module F, returning the application to a consistent state from module F's point of view. It then calls F's error recovery routine.

2.B Communication Rollback

While the above algorithm successfully rolls back application state, it does not take care of any MPI communication that process p performed since its last call to F. We can deal with this issue at the application-level with no additional support from the MPI implementation. When process p exits its call to module F it starts recording the communication operations that it performs on any of its communicators (just the type of operation and its arguments). When p is notified of the error, it will use the out-of-band notification mechanism to invoke callbacks on each process q that it communicated with since its last call to module F. p also sends to each q the portion of p's operations log that involves q. q can now compare this log to its own log to identify the operations that p has already performed that q has not yet performed and performs operations that match p's performed operations (i.e. if p called MPI_Send(q), q can call MPI_Recv(p) with a dummy buffer), thus freeing MPI's internal state associated with these operations. It then notifies all of its recent communication partners, including p, of its own operations log. This is similar to the Chandy-Lamport checkpoint protocol. If q was among the set of processes that previously called module F, it then uses its incremental checkpoint log to roll back to immediately after the call to F and then calls F's recovery procedure. Otherwise, it rolls back to immediately before its call to module R and resumes execution.

Note that this procedure only works if the operation does not perform externally visible operations such as printf or file writes. If this is not the case we will need to use additional logging mechanisms and place constraints on module recovery procedures.