MPI Device-Level Asynchronous Quiescence Interface

Mike Heffner

Evergrid, Inc.

June 5, 2008

v1.0

Overview

To checkpoint and restart an MPI application requires capturing the state of the messaging fabric between processes in a distributed application. The captured network state must represent an accurate snapshot in time of the messaging state such that the *global consistency* of the application is maintained. The common approach is to record logs of messages between senders and receivers and replay any unacknowledged messages during a restart. This approach works for applications that perform infrequent communication or exchange small – fast to copy – messages over high-latency fabrics. However, the overhead of this approach makes it impractical when communicating over high-speed, zero-copy RDMA-based fabrics.

The common approach taken for checkpointing message state on high-speed fabrics is to silence, or quiesce, the network fabric while a checkpoint is recorded. All in-flight messages are flushed to either the sender or receiver side and will be checkpointed as acknowledged.

This document proposes an interface to quiesce the communication fabric of an MPI job. The interface is proposed at an internal device or library-chosen implementation level. This level was chosen to decouple the operation from the normal MPI messaging. This allows the quiescence operation to be invoked asynchronously from the MPI application, typically from an external checkpoint/restart framework. The large benefit is that this approach does not require the MPI application to be rewritten to take advantage of checkpoint/restart. Additional quiescence interfaces could easily be added at higher levels (ie., the MPI_ interface level) that invoke the device-level quiescence interface.

The proposed interface is by no means complete. It is described in a manner that should facilitate a discussion towards a final version.

Note: This latter approach to quiescence was taken from Evergrid's Availability Services implementation of CP/R for the RDMA Mvapich stack. It is further described in “DejaVu: Transparent User-Level Checkpointing, Migration, and Recovery for Distributed Systems” published at IEEE International Parallel and Distributed Processing Symposium, 2007.

API

int MPID_Quiescence_enter(...)

This call informs the messaging fabric that it should enter a quiescent state. When this call returns successfully, all messages from and to this process must be flushed and additional communication to or from this process should be stopped. The parameters are currently left unspecified as there could be a use in specifying how the quiescence should happen – hard or soft tear down of communication. The return value of this function should be zero for a success and not zero on failure. Additional MPI calls may block if called during a quiescent state.

This call be invoked asynchronously during normal MPI communication, including from within the same thread context (ie., a POSIX signal handler). It is assumed that the state of the process is such that it could be successfully checkpointed and restarted.

TBD: Does this call require an OOB barrier operation across processes participating? Or is it the job of a checkpoint/restart system to perform a barrier after each process returns from this function?

int MPID_Quiescence_leave(int is_restarting)

This call will leave the quiescent state. Communication will resume between this process and any others that have also left the quiescent state. The *is_restarting* parameter is a boolean flag that informs the call whether the stack is resuming after a restart or not. Additional steps may have to be taken if the process is resuming from a restart – such as reinitializing communication libraries.

Undecided

Given that these APIs can be called asynchronously during normal MPI operations, there must be a way to temporarily block or disable the asynchronous quiescence calls. This would allow certain "critical sections" to be constructed around modifications of internal data structures used by both MPI communication and the quiescence operations.

Additional APIs will likely be required to correctly register call backs with the MPI library to block and unblock these asynchronous calls. These will be added in a future revision of this document.