Design and Implementation of Dynamic Process Management for Grid-enabled MPICH

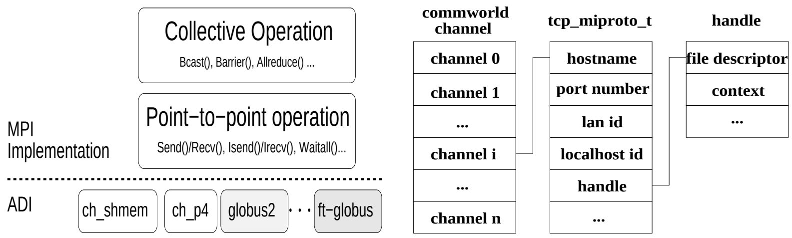
Sangbum Kim[1], Namyoon Woo[1], Heon Y. Yeom[1],
Taesoon Park[2], and Hyoungwoo Park[3]

[1] School of Computer Science and Engineering,
Seoul National University,Seoul,151-742,KOREA
{ksb,nywoo,yeom}@dcslab.snu.ac.kr
[2] Department of Computer Engineering,
Sejong University,Seoul,143-747,KOREA
tspark@kunja.sejong.ac.kr
[3] Supercomputing Center,KISTI,Taejon,Korea
hwpark@hpcnet.ne.kr

Abstract. This paper proposes the design and impementation of MPI_Rejoin() for MPICH-G2, which is not specified in the original MPI. The *'rejoin'* operation allows the restored process to rejoin the existing group by updating the corresponding entry of the channel table with the new physical address. This is the primitive of fault tolerance or task migration for message passing processes. The *'rejoin'* events are informed by hierarchical process managers. We have analyzed communication mechanism of MPICH-G2 and mesured the *'rejoin'* cost with NPB application(LU).

1 Introduction

The grid has attracted attention for its ability to utilize ubiquitous computational resources with a view of single system. The most of grid-related studies have concentrated on the static resource management. The dynamic resource management, especially the fault tolerance, has not been discussed enough in the grid research area. Several implementations of the fault tolerance have been proposed, which exploits the dynamic process management of PVM or LAM-MPI. The fearure of dynamic management is benefitted by the indirect communication method where the messages are transferred via intermediates like daemons. Channel reconstruction is simply accomplished by connecting the local daemon and the increase of message delay is inevitable. MPICH-G2 proposed by Argonne National Laboratory [1] is a grid-enabled MPICH that runs on Globus middleware. Each MPI process is aware of the physical channel information of the other processes hence it can send messages to the target processes directly. Once the process group set is initialized, no additional process can join the group. Assume that the failed process has been restored, it is assumed as an another instance at the view of the existing processes. After all, the processes do not invite the restored process to their group. It has been regarded as a tough issue to implement of the channel reconstruction for MPICH.



(a) Architecture of the MPICH (b) MPI_COMM_WORLD

Fig. 1. Architecture of the MPICH and Description of MPI_COMM_WORLD

In this paper, we propose the design and implementation of `MPI_Rejoin()`, which is not specified in the original MPI. The proposed operation allows the restored process to rejoin the existing group by updating the corresponding entry of the channel table with the new physical address.

2 Communication Mechanism of MPICH-G2

The communication of MPICH-G2 is based on non-blocking TCP sockets and active polling. In order to accept a request for channel construction, each MPI process opens a *listener port* and registers it on the file descriptor set with callback function to poll it. On the receipt of a request, the receiver opens another socket and accepts the connection. Then both two processes enter the *await_instruction* state where they exchange the metadata of the connection point. During this channel construction, the process with the larger rank ID plays a master role whether it is a sender or a receiver. Every process is aware of the listener ports for all processes after the initialization. The initialization in MPICH-G2 consists of two main procedures: the rank resolution and the exchange of listener information. Rank is the logical process ID in the group that is determined through the check-in procedure. After check-in procedure, the master process with rank 0 becomes aware of the processes' ranks and broadcasts it. Then, each process creates the listener socket and exchanges its address with the siblings processes to construct the `MPI_COMM_WORLD`. Figure 1(b) presents `MPI_COMM_WORLD`, the data structure containing the listener and channel information for all processes.

3 MPICH-GF

MPICH-GF [2] is our own fault tolerant MPICH implementation on grids that is originated from MPICH-G2. It supports the coordination checkpointing, the sender-based message logging and the receiver-based optimistic message logging for guarantee the consistent recovery. We have implemented those recovery algorithms on the basis of the user-level checkpoint library and the process rejoining module described in Section 4. Our MPICH-GF implementation is accomplished at the virtual device level only and it does not require any modification of application source code or the MPICH sub-layer upon ADI. MPICH-GF is aided by the hierarchical management system which is responsible for the failure detection and the process recovery.

global rank (4 Bytes)	hostname (256 Bytes)	port number (10 Bytes)	lan id (4 Bytes)	localhost id (40 Bytes)	fd (4 Bytes)

Table 1. Rejoin message format

4 Design of `MPI_Rejoin`

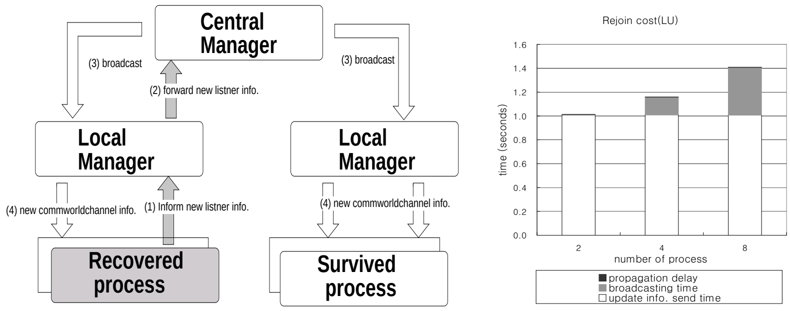
We've devised `MPI_Rejoin()` function in order to allow the recovered process to be recognized as the previous process instance. MPICH-G2 has many reference variables related to the system level information. These variables are used by select() system call for non-blocking TCP communication. The read / write callback functions also are stored in these variables. Since these sytem level information are not valid any more in a new process instance, they should be re-initiated in order to prevent a recovered process from accessing the invalid file descriptors. Then, the new physical address of the restored process should be notified to the other processes. Figure 2(a) shows the rejoin protocol incurred by `MPI_Rejoin()`. When a process failure is detected, a failed process is recovered with latest checkpoint image by local manager and call the `MPI_Rejoin()` function. `MPI_Rejoin()` function execute the following sequences.

– clearing the reference variables related to the system level information.
– reopening its listener port.
– sending its new channel information (Table 1) to the local manager.

Then the local manager forwards this report to the central manager. The central manager broadcasts this update report to the all local managers. On the receipt of the update report, local manager send the channel information by using message queue and SIGUSR1 signal to its processes. All survived processes modify the corresponding entries of their `MPI_COMM_WORLD` and set to NULL the value of *handle*. This makes the survived processes to access the failed process's new listner port in order to reconstruct the channel. This operation don't need to change the application source code. We have limitations that this operation don't support MPI-2 and concurrent failure.

5 Experimental Results

The *'rejoin'* cost per single failure is presented in Figure 2(b). *'update info. send time'* is the time at a failed process from re-lauching and reading the checkpoint file from hard disk to sending the update channel information to local manager . Since this cost is not influenced by the number of processes which participate in the current working group, it shows nearly constant time. We have measured *'broadcasting time'* at central



(a) Protocol between Manager and Processes

(b) Rejoin Overhead

Fig. 2. Protocol and Performance result

manager from receiving the a failed process's update channel informatioin to receiving the acknowledgement from all proces in current working group. Figure 2(b) shows that *'broadcasting time'* increase over the number of processes. *'propagation delay'* is pretty small and almost constant.

6 Conclusion

We have presented the design and implementation of process rejoining for MPICH on grids. The proposed operation is the primitive of fault tolerance or task migration for message passing processes. We expect that the on-demand channel updating will be more efficient since each parallel process tends to communicate with specific adjacent processes. On-demand in this context means that instead of broadcasting the new channel information, process asks the update only when it is necessary.

References

1. I. Foster and N. T. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of SC 98*. ACM Press, 1998.
2. N. Woo, S. Choi, H. Jung, J. Moon, H. Y. Yeom, T. Park, and H. Park. MPICH-GF: Providing fault tolerance on grid environments. In *The 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2003), the poster and research demo session*, MAY 2003.