An MPI failure detector over PMPI[1]

Donghoon Kim

Department of Computer Science, North Carolina State University

Raleigh, NC, USA

Email : {dkim2}@ncsu.edu

Abstract

Fault Detectors are valuable services which provide information about process failures in large-scale parallel systems. Previous many studies suggest guidelines for the implementation of a fault detector. However, a practical approach to implementation is another challenge due to various parallel system environments of both hardware and software. This study explains that Fault detector is able to be transparent, scalable, and portable. The experimental results show that Fault Detector can be embedded to any MPI application with negligible overhead.

1. Introduction

Modern scientific applications on Massive Parallel Processing Systems have execution times ranging from day to months. These long-running MPI applications on clusters are prone to node or network failures as the systems scale[1]. The MPI application may have no progress in the case of node or network failures if such an application needs to exchange its computation results through the communication. Furthermore, the recovery overhead would be increased unless the fault detection services provide timely detection. On the other hand, the overhead of fault detection would be increased as the frequency of fault detection increases for monitoring accurate failures. Thus, the Fault Detector still provides valuable services which are system management, load balancing, and replication as well as failure detections.

Previous many studies suggest guidelines of theoretical methodology for the implementation of Fault Detection services. However, a practical approach to implementation is another matter because various parallel system environments of both hardware and software yield more complicated other issues due to the property of unreliable failure detectors, that is, completeness and accuracy[2].

[1] The purpose of this paper is to complete CSC548 Parallel Systems - Section 001 supervised by Dr. Frank Mueller in Computer Science, North Carolina State University, Fall 2009.

Assume that the system model provides certain temporal guarantees on communication or computation called partially synchronous [3], the Fault Detector (FD) is able to utilize time based scheme, namely, ping-ack based implementation with the following proprieties :

- Transparency – The FD is launched in MPI_Init routine with a MPI profiling interface, which creates FD threads. The FD runs independently with a unique communicator different from an application program. When MPI application program pass through MPI_Init, FD is also running on each processes without additional touch.

- Scalability – The FD sends a check message sporadically at any time when an application program has a routine to communicate. It would not lead to high communication overhead compared with the frequency of periodic check message since the FD at each node avoids redundant check messages for a defined time period.

- Portability – MPI application can be compiled with FD if the user just adds FD source code in the same directory before compiling MPI application source codes.

In this paper, I implement the Fault Detector over MPI profiling layer to detect a failure of MPI application or network. The experimental results show that the Fault Detector has the above proprieties with the negligible overhead since I use sporadic communication approach.

The rest of this paper is structured as follows. Section 2 describes the design of Fault Detector and practical methods. Section 3 discusses implementation issues. Section 4 is experimental framework. Section 5 demonstrates the experimental results and analyzes the output. Section 6 reviews the related work of Fault Detection service. Section 7 is the conclusion and the future direction of this work.

2. Design

Figure 1 shows the diagram of the fault detector for sporadic communication which I have implemented so far. The arrows present the function call from MPI applications to Fault Detector. Suppose that the MPI application such as NAS Parallel Benchmark (NAS PB) runs on Massive parallel processing (MPP) environment. As the most MPI tools utilize, MPI profiling layer (PMPI) intercepts MPI calls during application execution. When MPI_Init is called in the application, Fault Detector (FD) is also launched since the code for calling FD is inserted in MPI_Init. FD code has been implemented with C code while some of NAS PB has been written by Fortran code. Thus, the Wrapper Function (WF) is for the application written in Fortran at which WF links MPI functions of the application to PMPI.

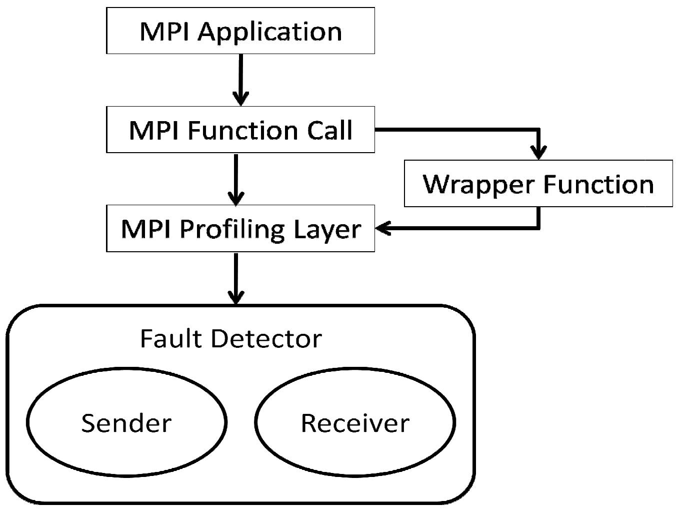


Figure 1. The Diagram of Fault Detector

Upon running FD, FD executes its own separate routine independent from the application. The new communicator, FD_COMM is used for FD communication routines to provide independent execution among FDs on MPP environment. FD consists of two threads, FD sender and FD receiver. Whenever MPI communication routines (e.g. MPI_Send or MPI_Isend) in the MPI application program are called, the corresponding PMPI routines are also executed. Each PMPI routine has three steps such as pre-processing, PMPI function call and post-processing. As an example of MPI_Send, Pre-processing registers a destination node with current time in Queue and sends a signal to FD sender in case of waiting for a signal since Queue is empty. PMPI function call executes normal function like PMPI_Send for the application execution. Post-processing deletes the destination node registered in Queue if returned SUCCESS in PMPI_Send. FD manages the status of neighbor nodes with Queue which is implemented using a doubly liked list with node ID, timestamp, check-in and so on. The FD is a thread created by pthread_create() function which works independently side by side with the application program. The FD uses two messages, ALIVE and ACK. ALIVE message is to check whether a destination node is alive or not. ACK message is to verify from a destination node. The FD should consider the time delay between two nodes with communication and computation time. The FD could suspect a destination node to be failed if no ACK message is received in correspondence to ALIVE message. The FD should be integrated into MPI environment. The following is the more detailed description of both FD sender and FD receiver.

- FD sender : It is supposed to send the ALIVE message unless Queue is empty. Before sending ALIVE message, FD sender checks the timestamp of a destination node in Queue whether the delay time passes the timestamp or not. The purpose of the delay time is not to make a redundant message. If delay time passes the timestamp, FD sender sends ALIVE message to that destination node and then flips check-in flag to indicate that FD already sent ALIVE message and is waiting for ACK message from that node with updated timestamp. When FD rechecks this node for the next turn, FD sender is able to suspect this node as a node failure if this node still exists in Queue. FD sender sorts Queue by updated timestamps in ascending order after one cycle.

- FD receiver : It is supposed to receive either ALIVE or ACK message. FD receiver probes periodically whether a message is arrived or not. On receiving a message, FD receiver takes the next action according to MPI_TAG. FD receiver replies back ACK message for ALIVE message while deleting a node ID from Queue for ACK message.

3. Implementation Issues

I have implemented three kinds of FDs depending on algorithms such as periodic all-to-all, and tree structure and sporadic FD. In this paper, I only show sporadic FD since it is more reasonable and practical approach regarding the performance.

FD utilizes pthread wait and signal to run continuously. The signals are conveyed to FD sender whenever MPI communication routines are called. FD sender keeps working as long as Queue has a destination node. FD sender also checks queue regularly such as every 20ms unless there is a signal. Queue is updated whenever there is any change such as insert, delete, timestamp update and etc. Queue is maintained by many internal functions in FD which requires consistent changes so that Queue is the critical section controlled by pthread lock and unlock.

FD should keep running until MPI applications terminate. FD might make processor keep busy so that it causes the performance of MPI application to go down. Thus, FD goes to sleep for some time after one cycle in both FD sender and FD receiver. In this implementation, 20ms is the sleep time.

The CG benchmark is written in Fortran so that WF is called whenever MPI routine executes. Fortran compilers are different, that is, one of the following function name is used for MPI_Send as an example, mpi_send_, mpi_send__, MPI_SEND, or MPI_Send. *mpi_send_* is used on opt10. The all arguments are pointer arguments in WF. Furthermore, the *ierr* argument at the end of the argument list in Fortran is not used in C because the ierr is an integer and has the same meaning as the return value of the routine in C.

4. Experimental Framework

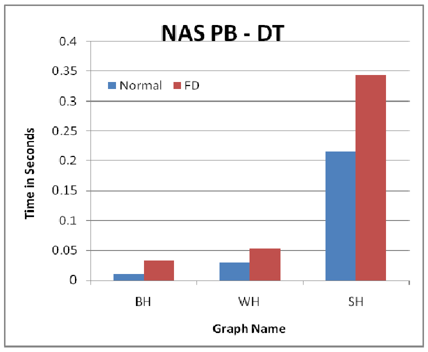
I conducted my performance evaluations on a local cluster. All machines are 2-way SMPs with dual-core AMD Opteron 265 processors by a Gigabit Ethernet switch running Fedora Core 5 Linux x86 64 with MPICH2 for FD test[4]. I used 3 nodes as an experimental cluster for limited resource.

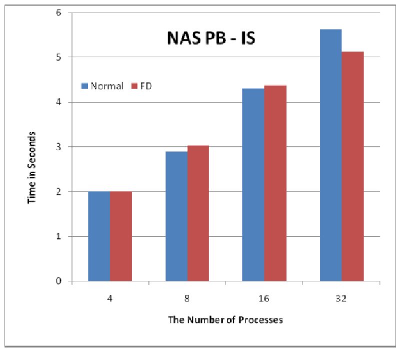
5. Experimental Results

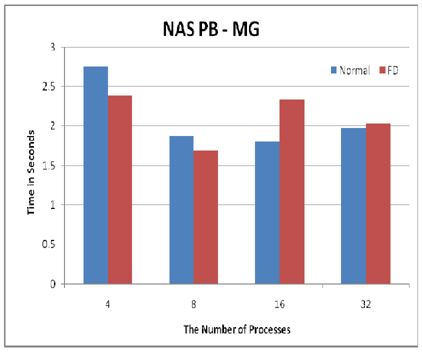
In this section, experimental results show how FD affects the performance of MPI applications. FD has been tested with the CG in NAS Parallel Benchmark (NPB) suite as a basic test. I still have several implementation issues on several test cases related Queue management by pthread lock & unlock. I make minor changes in PMPI routines removing FD processing codes in case of some issues exists. However, all the basic tests in the graph do not have any issue. Thus, the experimental results ensure the confidence for all cases.

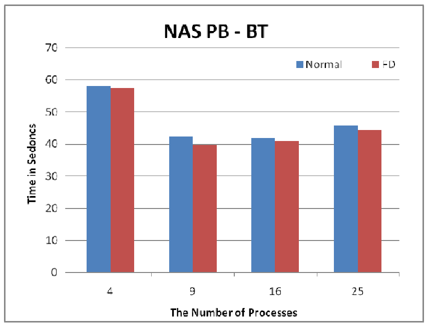
5.1 Performance overhead

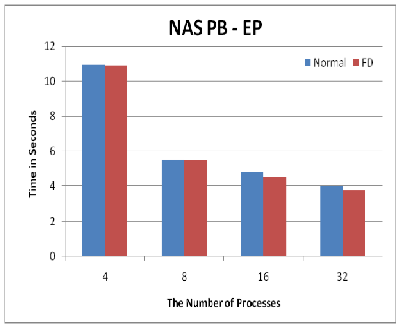
Figure 2 shows the performance results of NPB in comparison of NPB with FD. Normal in figure indicates the performance of NPB while FD indicates the performance of NPB with FD code. X-axis presents the number of processes and Y-axis represents the time in seconds in NPB output.

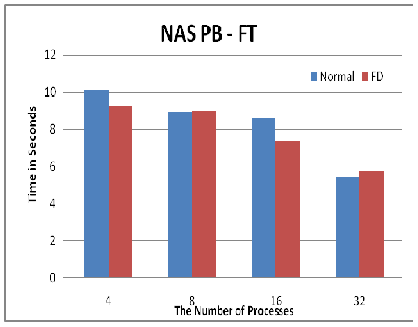












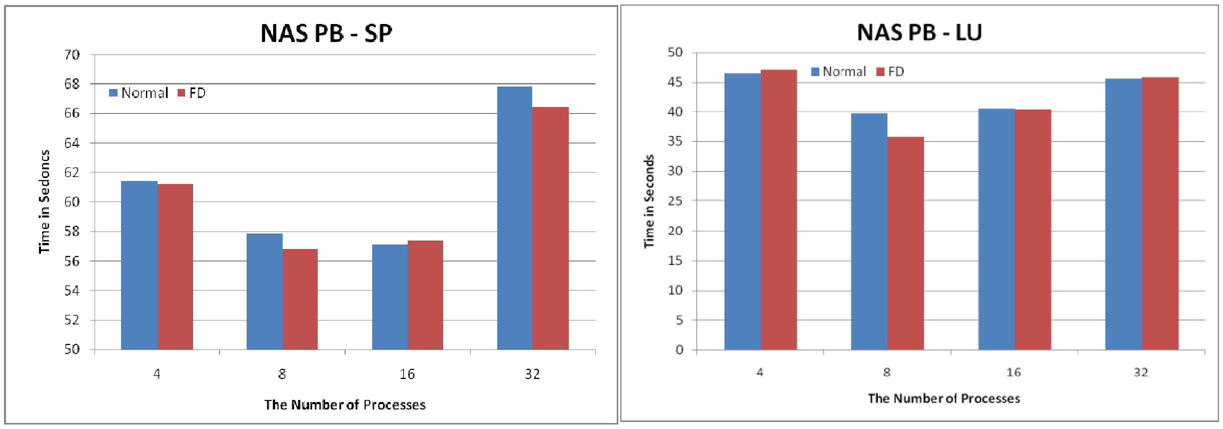


Figure 2. The performance results with FD

DT and IS in NPB are written in C while the others are written in Fortran so Fortran NPBs call additional wrapper functions. In DT, right bar (red color) is higher than left (blue color) bar because the execution time in DT is too short and relatively communication time such as round-trip time is higher than computation. However, we can say that there is no overhead to call wrapper function and there is no significant difference between C and Fortran. Three steps in MPI communication routines affect the overall performance Overwhelming trend in Figure 2 indicates that there is no performance difference between Normal and FD.

5.2 Communication overhead

Another overhead is communication overhead which is also able to be negligible. Because each MPI communication routine has three steps as I mentioned in Introduction. Post processing is to remove destination node in Queue if SUCCESS returned. This means that FD may send ALIVE signal to destination node rarely since a destination node is deleted as soon as it registers in Queue for most cases. I have tested many cases changing time parameters such as sleep and timedwait in order to make high communication overhead intentionally. However, it turns out less than 1% even in worst case. Periodic based algorithms such as all-to-all and tree structure may affect communication overhead because it should send and receive a signal periodically every a given time interval until MPI applications terminate. Thus, communication overhead in this work is able to be negligible in sporadic FD.

6. Related Work

In [2], Tushar and Sam classify 8 classes of failure detectors by specifying the completeness and accuracy properties and show how to reduce 8 failure detectors to 4 and how to solve consensus for each class by defining consensus problem. This paper affects many contemporary papers because it clearly indicates the false positive problem of many practical systems such as asynchronous system.

In [3], Srikanth and et al. address celerating environments due to a system upgrade or periods of high stress where absolute time speeds could increase or decrease. Bichronal timer with the composition of action clocks and real-time clocks is able to utilize in celerating environments. My implementation is only for real-time clocks at local node.

In [5], Stephane and et al. implemented Fault Detector in P2P-MPI environment with heartbeat counter. This paper addresses failure information sharing and consensus phase. They mention fault detection overhead because they send heartbeat periodically. It is practical approach in commodity system.

7. Conclusion

In this work, I implement sporadic Fault Detector based ping-ack messages. There are still some implementation issues. However, I can say the practical Fault Detector is able to be implemented with the following properties, transparency, scalability, and portability. The experimental results show that Fault Detector has negligible overhead for both communication and performance.

I should add the global view list for node failures and how to consensus the difference among different global view lists. I will implement this feature in my future work [6].

8. References

[1] Jitsumoto, H., Endo, T., Matsuoka, S., "ABARIS: An Adaptable Fault Detection/Recovery Component Framework for MPIs," *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)* pp.1-8, March 2007.

[2] Tushar Deepak Chandra , Sam Toueg, Unreliable failure detectors for reliable distributed systems, Journal of the ACM (JACM), v.43 n.2, p.225-267, March 1996

[3] Srikanth Sastry, Scott M. Pike , Jennifer L. Welch "Crash fault detection in celerating environments" *IEEE International Parallel and Distributed Processing Symposium (IPDPS 2009)* pp.1-12, 2009.

[4] http://moss.csc.ncsu.edu/~mueller/cluster/opt/

[5] Evaluation of Replication and Fault Detection in P2P-MPI, St´ephane Genaud, Choopan Rattanapoka, IPDPS09

[6] http://www4.ncsu.edu/~dkim2/csc548/csc548.htm