병렬 컴퓨터에서의 결함 허용 메시지 전달 인터페이스 구현

(An Implementation of Fault-Tolerant Message Passing Interface on Parallel Computers)

송 대 기 † 이 철 훈 ††
(Dae-Ki Song) (Cheol-Hoon Lee)

요 약 메시지 전달 인터페이스(MPI)는 기존의 다양한 병렬 프로그램 개발 환경을 표준화한 것으로써, 메시지 전달 인터페이스를 기반으로 하는 병렬 컴퓨터 시스템은 응용 프로그램을 수많은 프로세서들에 분산 배치시켜 수행한다. 구성되는 각각의 프로세서 노드들은 연산을 하고 서로 결과를 메시지로 교환하여 수행을 하게 된다. 그러나 병렬 컴퓨터를 구성하는 노드들 중에서 어느 한 노드 또는 작업 중인 프로세스가 고장을 일으킨다면 수행되는 응용 프로그램은 그 동안의 수행 결과를 잃게 되며, 또한 응용 프로그램을 구성하는 모든 프로세스들은 중단될 것이다. 본 논문에서는 이와 같은 문제를 해결하기 위해 기존의 MPI에 고장 관리자(Fault Manager) 모듈을 추가함으로써 고장 허용 메시지 전달 인터페이스인 FT-MPI를 제안한다. 제안한 FT-MPI는 고장 처리를 위한 추가적인 하드웨어 지원이 필요하지 않으며 기존의 MPI 응용 프로그램들이 수정 없이 수행될 수 있다는 장점을 지닌다. 제안한 고장 허용 방법은 프로세스 이중화 기법인 hot-spare 방법을 사용하였으며, 시뮬레이션을 통해 제안한 FT-MPI가 고장이 발생하더라도 응용 프로그램이 올바르게 수행되며, 고장 허용 기능으로 인한 수행 시간상의 오버헤드는 5%를 넘지 않음을 보인다.

Abstract The Message-Passing Interface(MPI) is a standard interface for parallel programming environment, based on that application programs run on the processors of a parallel computer. Processor nodes execute processes consisting the program by passing messages to one another. During executing, however, if a fault occurs on a processor node or a process, this will result an inconsistent state, and consequently, the whole program will have to be stopped. To solve this problem, in this paper, we propose a fault-tolerant message passing interface(FT-MPI) by adding a fault manager module to MPI. The proposed FT-MPI does not need any hardware support, and each application program based on MPI can run on the FT-MPI without any modification. The proposed fault tolerance scheme uses the so-called hot-spare process duplication method, and verified by simulations that application programs run despite of any fault with less than 5% overhead on execution time.

1. 서 론

수십 혹은 수백 개 이상의 프로세서들로 구성된 병렬 컴퓨터는 주로 기상 관측이나 석유 탐사 등과 같이 과학 계산용 응용 프로그램을 수행하는 용도로 사용이 되고 있으며, 이들 응용 프로그램들은 방대한 양의 데이터를 처리하며 또한 수행 시간이 많이 소요되므로 여러 프로세서들에 분산되어 서로 통신을 하면서 병렬로 수행된다[1]. 이 경우 응용 프로그램을 수행하는 특정 프로세서에 고장이 발생하게 되면 전체 프로그램이 비정상적으로 중단되며, 고장난 프로세서를 제외한 다른 프

· 본 연구는 과학기술부-과학기술정책관리연구소 주관 미래 원천 국책과제의 연구비에 의하여 연구되었음.
† 비 회 원 : 충남대학교 컴퓨터공학과
dksong@comeng.chungnam.ac.kr
†† 정 회 원 : 충남대학교 컴퓨터공학과 교수
chlee@comeng.chungnam.ac.kr
논문접수 : 1999년 4월 14일
심사완료 : 2000년 3월 3일

로세서들 상에서 처음부터 다시 재 수행하여야 하므로 이로 인한 경제적 시간적 손실이 크다고 할 수 있다. 특히 기상 관측 등과 같은 실시간성을 요구하는 응용 프로그램의 경우에는 아주 심각한 결과를 초래할 수 있다. 이와 같은 고장으로부터의 손실을 줄이기 위해서는 시스템의 신뢰성과 가용성을 높여야 하며, 따라서 이들 응용 프로그램을 수행하는 병렬컴퓨터에서는 고장 허용 기능이 필수적이다[2].

본 논문에서는 병렬컴퓨터 상에서의 메시지 전달 방식의 표준인 메시지 전달 인터페이스(MPI: Message Passing Interface)에[4,5,6] 고장 허용 기능을 추가한 고장 허용 메시지 전달 인터페이스(FT-MPI: Fault-Tolerant Message Passing Interface)를 제안한다. 제안한 FT-MPI는 병렬컴퓨터에서 고장 처리를 위한 별도의 하드웨어 지원이 필요하지 않으며, 또한 MPI를 이용한 기존의 모든 응용 프로그램들도 수정 없이 FT-MPI 상에서 고장 허용성을 가지며 수행이 되는 장점을 지닌다. 제안한 FT-MPI는 고장 허용 기능을 제공하기 위하여 프로세스 이중화 방법을 사용한다.

프로세스 이중화 방법은 고장에 대비하여 프로세스를 쌍으로 (즉, 주 프로세스와 백업 프로세스) 중복시키는 것으로[2,3,7], 프로세서 자원의 중복과 프로세스의 컨텍스트(context)의 보존 측면에서 hot-standby[9]와 hot-spare[7] 방법으로 나누어진다. Hot-standby 기법은 실제 프로그램 수행은 주 프로세스에서만 이루어지며 고장에 대비하여 자신의 상태를 주기적으로 백업 프로세스에게 체크포인팅(checkpointing)한다[8,13]. 이 경우 주 프로세스에 고장이 발생하면 백업 프로세스는 가장 최근의 체크포인트 정보를 이용하여 롤백(rollback)하여 그 시점부터 수행을 재개한다[10]. 이 방법은 프로세스 이중화를 위하여 프로세서 자원을 많이 사용하지 않는다는 장점이 있지만, 주기적인 체크포인팅을 위한 실시간 부하가 많은 점과 고장 복구 과정이 복잡하고 느리다는 단점이 있다. 다음으로 hot-spare 기법은 주 프로세스와 백업 프로세스가 동시에 동일한 작업을 처리하도록 하는 것으로써 이중화 된 두개의 프로세스 중에서 어느 하나에 이상이 발생하더라도 나머지 프로세스에 의해서 그 작업이 지속적으로 수행 될 수 있도록 하는 기법이다[14]. 주 프로세스와 백업 프로세스가 동시에 동일한 작업을 수행하게 되므로 두 배의 프로세서 자원이 소요되고, 서로 이중화 된 두 개의 프로세스 사이에 동기화 문제가 발생한다는 단점이 있지만 백업 프로세스를 통해 바로 복구가 이루어지므로 빠른 복구 기능이 가능하며 주기적인 체크포인팅(checkpointing)을 하지 않으므로 실시간 부하가 없다는 장점을 가지고 있다.

또한 기존의 병렬 응용 프로그램의 고장 허용을 위한 언어인 FT-Linda는[15] 병렬 프로그래밍 언어에 고장 허용 기능을 추가한 것이다. FT-Linda는 고장 허용을 위하여 안전 튜플 공간(stable tuple spaces)과 튜플 공간의 연산에 대하여 단일 수행(atomic execution)을 제공한다. 전자의 경우 데이터 손실에 대한 보호를 제공하고, 후자의 경우 고장의 영향이나 동시성에 대한 고려 없이 튜플 공간 연산을 수행할 수 있도록 한다. 그러나 FT-Linda의 경우 고장 허용을 위하여 시스템에서 안전 저장공간(Stable Storage)을 필요로 하며, 또한 수행의 일관성을 위하여 단일 수행을 보장해 주어야 하므로 시스템에 부하가 크다는 단점을 지닌다.

앞에서 설명한 바와 같이 병렬컴퓨터는 수많은 프로세서들로 구성 되어 있으며 실시간성과 고성능을 요구하는 과학 계산용 응용 프로그램을 수행하므로, 프로세서 자원이 많이 요구되지만 실시간 부하가 적고 고장 복구 시간이 적게 걸리는 hot-spare 방식이 적합하다고 본다. 따라서 본 논문에서 제안한 FT-MPI는 hot-spare 기법을 사용한다.

제안한 FT-MPI는 고장을 처리하기 위하여 기존의 MPI 라이브러리에 고장 관리자(FTM: Fault Manager) 모듈을 추가하였으며, 이 모듈이 MPI 상에서 고장 탐지, 진단 및 복구 등의 고장 허용 기능을 수행하도록 하였다. 또한 고장 관리자는 hot-spare 기법을 적용하기 위해 본문에서 설명될 여러 기능들을 처리한다.

본 논문의 구성은 다음과 같다. 제 2 장에서는 제안한 FT-MPI에 추가된 모듈인 고장관리자가 고장을 처리하는 여러 가지 고장 허용 기능들에 대해서 기술하고, 제 3 장에서는 고장 복구 시 처리해 주어야 하는 일관성 유지 문제를 설명한다. 제 4 장에서는 FT-MPI 상에서의 고장 복구 실험과 그 결과를 보여 주며, 마지막으로 향후 연구 과제에 대한 내용을 기술하고 결론을 맺는다.

2. 고장 허용 메시지 전달 인터페이스 (FT-MPI)

기존의 메시지 전달 인터페이스를 기반으로 작성 된 병렬 응용프로그램은 구성중인 어느 한 프로세스의 고장으로 인해 전체 응용프로그램이 중단될 수 있다.

그림 1은 기존 MPI 기반 응용프로그램의 문제를 나타내는 것으로써, 그림 1(a)와 같이 P3 프로세서에서 고장이 발생한 경우 P3에서 수행 중인 프로세스의 수행

중단으로 인해 그림 1(b)와 같이 응용프로그램을 구성하는 모든 프로세스에 영향을 미쳐서 전체 응용 프로그램 수행 중단을 초래한다. 본 논문에서는 이러한 문제를 해결하기 위하여 기존의 MPI를 수정하여 프로세서에 고장이 발생하더라도, 이를 처리하여 응용 프로그램이 지속적인 수행을 할 수 있도록 하였다. 고장 허용 기능이 추가된 메시지 전달 인터페이스를 고장 허용 메시지 전달 인터페이스(FT-MPI)라 칭하며, 본 장에서는 제안한 FT-MPI의 구조와 고장 허용 기능에 대하여 설명한다.

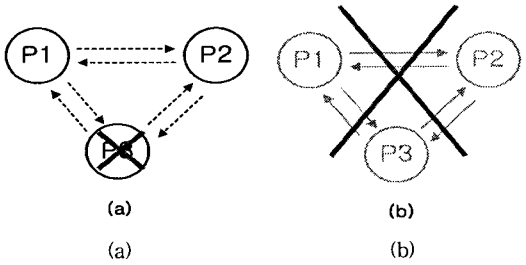


그림 1 기존 MPI 기반 응용프로그램의 문제점 (a)프로세서 P3 에서 고장 발생 (b)전체 응용프로그램의 중단

2.1 고장 관리자(Fault Manager)

고장 FT-MPI는 MPI 상에서 고장 허용 기능을 제공해 준다. 그림 2는 병렬 응용 프로그램 상에서 FT-MPI가 차지하고 있는 위치를 보여 주고 있다.

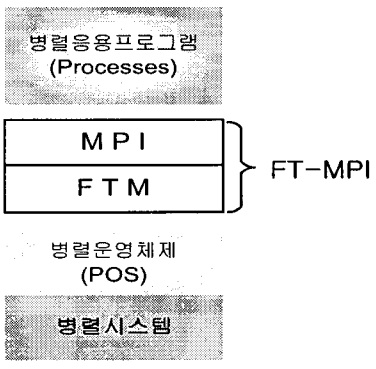


그림 2 FT-MPI의 위치

그림 2에서 보는 바와 같이 FT-MPI는 기존의 MPI에 고장 허용 기능을 제공하는 고장 관리자(FTM : Fault Manager)를 추가한 것이다. 고장 허용을 위한 기본 개념은 프로세스 이중화로써, FTM은 이를 위하여 병렬 운영체제와 MPI 중간에서 프로세스와 메시지의 이중화와 관련된 기능을 수행하며 다음과 같은 기능을 갖는다.

(1) 메시지 이중화

하위 계층인 운영체제로 전달되는 메시지를 받아, 메시지를 전달하고자 하는 주 프로세스뿐만 아니라 백업 프로세스에게도 동일한 메시지를 전달해 준다. 또한 주 프로세스와 백업 프로세스로부터 동일한 메시지를 받아서 상위 계층인 MPI로 전달해 주는 역할을 담당한다.

(2) 프로세스 이중화

기존 MPI 상의 단일 프로세스는 주 프로세스와 백업 프로세스로 이중화된다. 주 프로세스와 백업 프로세스는 서로 다른 프로세서에 생성이 되며, 두 프로세스 모두 같은 상태로 초기화된다. FTM은 이 과정을 자동으로 수행하며, 백업 프로세스에게 적당한 프로세스 식별자(Process-ID)를 할당하는데, 이들 프로세스 식별자는 프로세스의 주, 백업 관계를 구별하는데 사용된다.

(3) 고장 허용 기능

고장 허용을 위한 고장 탐지, 진단, 통보, 복구 기능을 수행하며, 각 부분의 자세한 사항은 다음에 설명한다.

(4) 복구 후 프로세스 일관성 유지

프로세스가 이중화되기 때문에 고장이 발생하였을 경우 주 프로세스와 백업 프로세스의 상태가 틀릴 수 있는데, 응용 프로그램의 정상 동작을 보장하기 위해선 복구 후 두 프로세스의 상태가 동일해야 한다. 본 논문에서는 이것을 일관성 유지 문제로 설명하며 자세한 내용은 제 3장에서 기술한다.

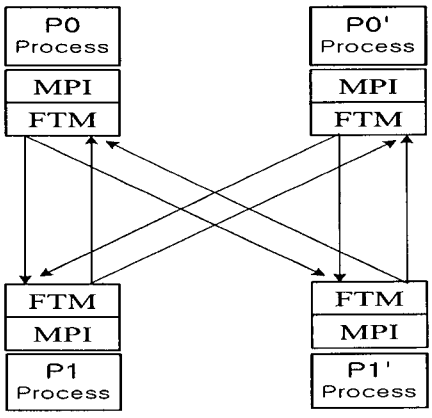


그림 3 FT-MPI 기반 하에서의 정상동작

2.2 정상 동작 과정

그림 3에서는 FT-MPI를 이용한 고장 허용 응용 프로그램의 구성을 보여 주고 있다.

여기에서 응용 프로그램은 두 개의 프로세스 P0 와 P1으로 구성되어 있으며, 각 프로세스는 자신의 백업 프로세스 P0′과 P1′을 가지고 있다. 프로세스의 고장이 없는 경우, 프로세스의 메시지는 FTM에 의해서 이중화 되어 상대방 프로세스들에게 전송되고, 수신시에도 이중화 된 메시지가 수신된다. P0 프로세스는 P1, P1′ 프로세스에게 메시지를 전송하고, 마찬가지로 P0′ 프로세스 또한 P1, P1′ 프로세스로 메시지를 전송한다. 메시지를 수신하는 P1, P1′ 프로세스의 FTM은 P0, P0′로부터 같은 일련 번호의 메시지를 확인하여 상위 계층의 프로세스에게 메시지를 전달하게 된다. 메시지의 전송이나 수신 완료 때까지 프로세스의 수행은 블록(block) 되는데, 메시지 수신 시 상대편 프로세스 쌍으로부터 메시지를 모두 받아야

프로세스는 수신 작업을 완료하고 메시지의 처리를 시작한다. 메시지 송신 완료는 상대편 프로세스 쌍으로부터의 ACK(Acknowledgement)신호로 결정된다.

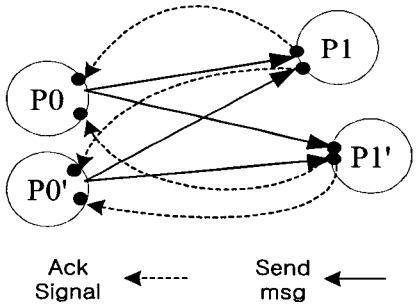


그림 4 메시지 전송 시 ACK 신호

그림4는 P0, P0′ 프로세스 쌍과 P1, P1′ 프로세스 쌍 사이에 메시지 교환이 이루어지는 과정이다. P1, P1′ 프로세스는 P0, P0′ 프로세스 모두에게서 메시지를 받을 때까지 블록 되고, 메시지를 수신한 후 ACK 신호를 전송한다. 메시지를 전송하는 P0, P0′ 프로세스 쌍은 P1과 P1′으로부터 ACK 신호를 받을 때까지 블록된다.

2.3 고장 탐지 과정

고장 탐지는 응용 프로그램을 구성하는 프로세스들 중 어느 프로세스에 이상이 있는지 발견하는 과정이다. 응용 프로그램을 구성하는 모든 프로세스들은 이중화로 중복되는데, 각각의 프로세스 쌍들은 메시지 교환 시 주 프로세스와 백업 프로세스 모두로부터 메시지를 전송하고 수신하게 된다. 만약 두 프로세스 중 어느 한 곳에서만 메시지가 도착하고 다른 곳에서 메시지가 도착하지 않으면, 고장 관리자는 고장이 발생한 것으로 간주한다. 또한 메시지 전송 시에도 어느 한 쪽에 문제가 발생했을 경우 마찬가지로 고장 발생으로 간주한다. 다음은 메시지 수신시와 송신 시 고장을 탐지하는 과정을 나타낸다.

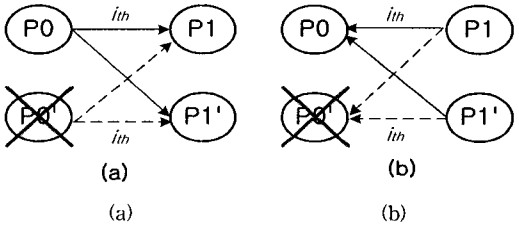


그림 5 고장탐지 과정 (a)메시지 수신 시 고장 탐지
(b)메시지 전송 시 고장 탐지

그림 5(a)는 메시지 수신 시, 그림 5(b)는 메시지 송신 시 프로세스의 고장을 탐지하는 그림으로 P0′ 프로세스에 고장이 발생한 경우이다. P1과 P1′ 프로세스는 상대편 P0, P0′의 프로세스 쌍으로부터 메시지를 수신해야 하지만, (a)의 경우 i번째 메시지를 P0로부터만 받게 되고, P0′ 프로세스로부터의 메시지 수신은 지정된 타임아웃(timeout) 시간 동안 메시지 수신을 하지 못하게 되므로 P1과 P1′ 프로세스의 FTM은 P0′를 고장으로 간주하고 P0′에 대하여 고장 진단을 수행한다.

2.4 고장 진단 과정

고장 진단 과정은 고장이 탐지 된 해당 프로세스를 검사하여 고장을 확인하는 과정이다. 고장을 탐지한 FTM은 고장난 프로세스의 짝 프로세스의 FTM에게 고장을 통보하고, 통보 받은 FTM은 짝 FTM에게 고장 진단 메시지를 보내어 일정 시간 동안 응답이 없으며, 고장을 확인하고 응용 프로그램이 관련된 다른 모든 FTM에게 고장 발생을 통보한다.

그림 6은 고장 진단 과정을 나타내고 있다. 프로세스의 고장을 탐지한 P1, P1′ 프로세스는 P0에게 이를 알리고, P0 프로세스의 FTM은 자신의 짝 프로세스 FTM에게 고장 진단 메시지를 보내는데, 응답이 있는 일시적 고장(transient fault)으로 간주하고 간단한 복구 과정을 거친 후 실행을 재개한다. 그러나 일정 시간 동안 응답이 없는 경우, 영구적 고장(permanent fault)으로 간주하고 응용 프로그램이 관련된 다른 모든 FTM에게 고장을 통보한다.

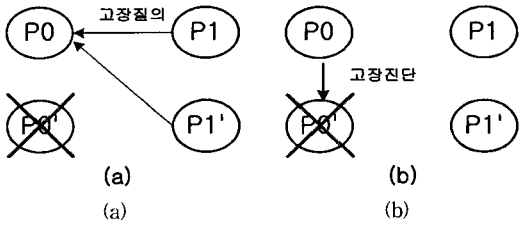


그림 6 고장진단 과정 (a)고장 질의 메시지 전송 (b) 고장 진단 메시지 전송

2.5 고장 통보 과정

고장 통보 과정은 고장 진단 후 해당 프로세스가 영구적 고장으로 판단되는 경우, 응용프로그램 상의 모든 FTM에게 고장 발생 사실을 알려 프로그램의 상태를 복구 모드로 변환하게 한다. 복구 모드에서는 모든 프로세스의 수행과 메시지 송수신 작업이 중단된다. 그림 7은 고장 통보 과정을 보인다.

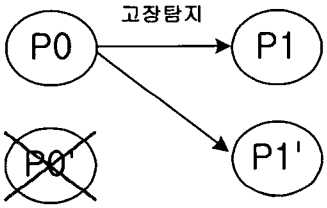


그림 7 고장통보 과정

2.6 고장 복구 과정

고장 복구 모드에서는 고장이 발생한 프로세스를 응용프로그램에서 제거한 후, 새로운 백업 프로세스를 다른 프로세서에 생성함으로써 고장을 처리하게 된다. 새로운 백업 프로세스의 생성은 쌍 프로세스 P0의 상태를 그대로 복사하는 방법을 사용하며, 이 두 프로세스는 재시작 시 동일한 상태에서 시작하게 된다. 이 과정에서 새로이 생성된 프로세스 P0''는 고장 전 프로세스 P0'의 상태와 차이가 있기 때문에 나머지 프로세스들과 일관성 유지에 관한 문제가 발생한다. 본 절에서는 복구 시의 수행 과정을 설명하며, 복구 시의 일관성 문제는 다음 장에서 설명한다.

새로 생성된 백업 프로세스는 새로운 통신 포트를 다른 프로세스들에게 알려 고장 복구 후 메시지 전송이 가능하도록 하며, 프로세스 식별자 등의 정보를 변경한다. 그림 8은 고장 복구 과정의 수행 내용을 보여준다.

고장 복구 과정에서는 모든 프로세스의 작업은 블록 되고 프로세스 P0의 FTM은 새로운 백업 프로세스를 생성할 프로세서를 선택한 후, P0 프로세스를 그대로 복사한 새로운 백업 프로세스 P0''를 만든다. P0의 FTM은 백업 프로세스에게 프로세스 식별자를 부여하고 필요한 정보들을 수정한다. 그림 8(b)는 새로운 통신 포트 정보를 전송하는 과정으로, 재시작 이후 프로세스간 통신을 위하여 새로 생성된 프로세스의 통신 포트로 정보를 갱신한다. 그림 8(c)는 모든 복구 작업을 마치고 프로세스가 재시작하도록 재시작 메시지를 전송하는 과정이다.

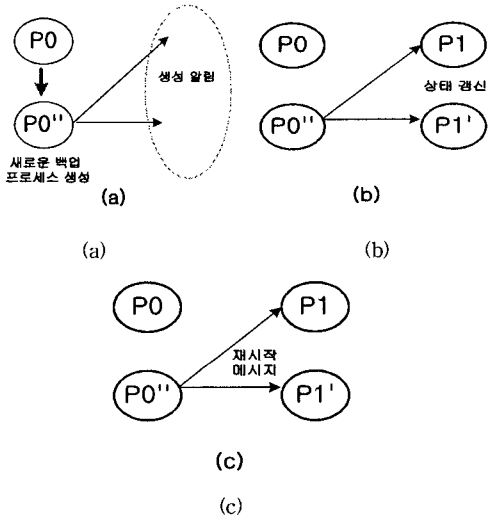


그림 8 고장복구 과정 (a) 새 백업 프로세스 생성 (b) 프로세스 상태 전송 (c) 재시작 메시지

3. 고장 복구 후 프로세스 간 일관성 유지

제안한 방법은 프로세스를 주 프로세스와 백업 프로세스로 이중화하여 같은 시간에 동일한 작업을 수행시키는 방법이다. 그래서 어느 하나의 프로세스에 고장이 발생했더라도 나머지 프로세스를 통하여 작업을 계속할 수 있게 된다. 하지만 고장 복구 과정을 수행할 때 새로운 백업 프로세스를 생성하여야 하는데, 이 과정에서 고장 전 상태와 고장 후 상태의 차이 때문에 프로세스들 사이에 동기가 맞지 않게 될 수 있다. 다음 그림은 이 문제를 설명한다.

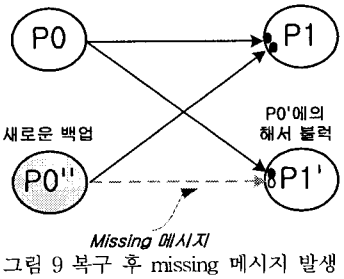


그림 9 복구 후 missing 메시지 발생

그림 9는 P0′ 프로세스에서 고장이 발생한 경우로써 P1′ 프로세스에서 고장을 탐지하고, 새로운 백업 프로세스 P0′′가 생성되면서 복구가 완료된 상태이다. 고장 복구 과정에서는 자신의 쌍 프로세스 P0를 사용하여, 백업 프로세스를 생성하게 된다. 여기에서 새로운 백업 프로세스 P0′′는 P0와 같은 상태를 갖게 된다. 하지만 고장 전 P0′의 상태와 새로 생성된 P0′′의 상태에서 차이가 발생하고, 메시지를 교환하는 P1′과 일관성이 깨지게 된다. 그림 9의 예에서는 P0′로부터 P1′프로세스로 보내어 지는 메시지가 missing 메시지가 된다. 이렇게 일관성이 깨졌을 경우, P1′프로세스는 P0′로부터의 메시지에 블록 되고 새로운 백업 프로세스가 생성되었음에도 불구하고 응용 프로그램은 정상 동작할 수 없게 된다.

3.1 시스템 모델

본 논문은 메시지 전달을 기반으로 하는 병렬 응용 프로그램을 대상으로 한다. 응용 프로그램은 다수의 프로세스들로 구성되며 고장 허용을 위하여 각각의 프로세스들은 주 프로세스와 백업 프로세스의 쌍으로 이중화되어 있다. 따라서 n 개로 구성된 응용 프로그램인 경우 내부적으로 이중화되어 있기 때문에 구성하는 전체 프로세스 수는 $2n$ 개가 된다. 복구 과정에서의 프로세스간의 일관성을 위하여 다음과 같은 사항을 가정한다.

[가정1] 주 프로세스와 백업 프로세스로 이중화 된 프로세스 쌍은 동시에 고장이 발생하지 않는다. 즉, 하나의 프로세서 고장(single fault)만을 가정한다. 또한 고장 복구 동안에는 또 다른 고장은 발생하지 않는 것으로 한다.

[가정2] 응용프로그램을 구성하고 있는 프로세스는 서로 다른 프로세서에 위치한다. 응용프로그램에 포함된 프로세스들은 서로 다른 프로세서에 위치시켜, 병렬 시스템의 컴퓨팅 노드에 고장이 발생하였을 경우, 동시에 여러 프로세스가 중단되는 일이 없도록 한다.

[가정3] 프로세스간 통신 채널에는 오류가 없다. 통신 채널의 오류에 의한 메시지 교환의 문제는 발생하지 않는다. 또한 메시지 교환시의 지연시간도 제한되어 있다.

가정1은 대부분의 상용 고장 허용 시스템에서처럼 단일 고장만을 허용하겠다는 것이고, 가정2와 3은 일반적으로 병렬컴퓨터는 수많은 프로세서들이 고속의 통신 채널로 연결되어 있다는 점을 고려하면 무리한 가정이라고 생각하지 않는다.

3.2 일관성 유지 방법

응용프로그램의 일관성이 깨지는 이유는 응용프로그램을 구성하는 프로세스들 사이에 missing 메시지나 orphan 메시지가 발생하기 때문으로[8], 복구 후 응용프로그램의 정상적인 수행을 위하여 missing 이나 orphan 메시지가 발생하지 않도록 해야 한다[16]. 본 논문에서는 프로세스 사이에 일관성을 맞추는 다음의 유지 조건을 두어, 고장 복구 과정에서 이 조건을 만족하도록 한다. 다음은 논문에서 사용한 표기에 대한 내용이다.

표 1 이 논문에서 사용되는 표기

표 기	설 명
$P1, P1', P2, P2', , Pn, Pn'$	프로세스
Pi'	Pi 프로세스의 프로세스 쌍
$\{Send,Destination,Seq\_num\}$ or $\{Recv,Source,Seq\_num\}$	메시지 이벤트 e (전송과 수신을 구분하고 목적지, 출발지 아이디, 일련번호로 나타냄)
$SENT_{i\to j}(e)$	Pi프로세스의 e 이벤트까지 Pi로부터 Pj프로세스에 보내어진 총 응용프로그램 메시지 수
$RECD_{i\to j}(e)$	Pi 프로세스의 e 이벤트까지 Pj로부터 Pi 프로세스가 받은 총 응용프로그램 메시지의 수

정 의1 : 어떠한 프로세스도 현재 시스템 상태에서 보내지지 않은 메시지나 송신 프로세스에서 미래의 수행동안 보내질 수 없는 메시지를 받지 않는다면, 시스템 상태는 일관성이 있다. 즉, 이것은 시스템의 프로세스들 사이에 orphan 메시지나 missing 메시지가 발생하지 않는 상태를 말한다.

정 의2 : 응용 프로그램의 수행 시간 t 의 이벤트의 셋을 컷(cut)이라 하고, 컷은 시간 t에서 프로세스의 수행상태를 나타낸다. 컷에는 프로세스 각각에 대하여 하

나의 메시지 이벤트만이 존재한다.

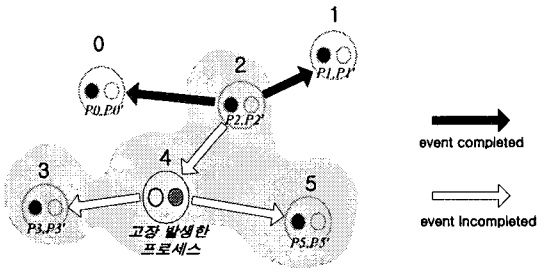


그림 10 시간 t 에서의 프로세스 상태

그림 10은 시간 t 에서의 프로세스 상태를 나타내고 있다. 구성되어 있는 프로세스들은 각각 하나의 이벤트를 가지고 있으면, 그 이벤트에 따라 상태가 표시된다. P0는 P2로부터 메시지를 수신 완료한 상태로, P4는 P3′과 P5′으로 메시지를 송신하며 또한 P2′로부터 메시지를 수신하는 상태로 정의할 수 있다.

정 리1 : 시간 t 의 컷 C에 포함되어 있는 임의의 모든 두 프로세스 Pi, Pj의 이벤트 쌍 ei, ej에 대해서 다음의 두 가지 조건 $\{SENT_i \rightarrow_j(e_i) = RECD_j \leftarrow_i(e_j)\}$과 $\{SENT_j \rightarrow_i(e_j) = RECD_i \leftarrow_j(e_i)\}$ 을 만족하면 컷 C는 일관성 있다.

증 명 : 오류가 없는 통신 채널 상에서 Pi 프로세스에 의해서 Pj프로세스에 전송된 메시지 수가 n 이고, 동일 채널 상에서 Pj 프로세스가 Pi 프로세스로부터 수신한 메시지 수가 m 이라고 가정한다면, 수신한 메시지의 수는 동일 채널을 통해서 전송 된 메시지의 수를 초과할 수 없기 때문에 $n \geq m$ 의 조건을 만족한다[9]. 또한 본 논문에서는 프로세스의 메시지 전송 시에 송신 측에서 수신 측의 ACK 신호를 받지 않으면 상태를 갱신하지 않도록 가정하였기 때문에 송신한 메시지 수는 수신한 메시지 수를 초과할 수 없으며, 따라서 $n = m$을 만족한다. 송신 메시지 수와 수신 메시지의 수가 동일하다는 것은 missing 이나 orphan 메시지가 없음을 뜻하며, 이는 두 프로세스간 일관성이 유지됨을 의미한다. 여기에서 n 과m 은 표 1에서 각각 $SENT_i \rightarrow_j(e_i)$, $RECD_j \leftarrow_i(e_j)$ 로 표현한 것이며, 따라서 $SENT_i \rightarrow_j(e_i) = RECD_j \leftarrow_i(e_j)$ 이며, 반대의 경우(전송 측이 Pj 이고 수신 측이 Pi인 경우)는 $SENT_j \rightarrow_i(e_j) = RECD_i \leftarrow_j(e_i)$ 을 의미한다. 그러므로 컷 C에 포함되어 있는 모든 쌍 이벤트에 대하여 위의 조건이 만족하면 일관성이 있다고 할 수 있다.

3.3 고장 처리 모드에서의 동기 과정

고장 처리 모드에서는 새로운 백업 프로세스가 생성된 후 자신과 메시지 교환이 있는 프로세스들과 일관성 조건을 검사하며, 새로 생성된 프로세스의 상태에 따라 각각의 프로세스들이 자신의 상태를 갱신하게 된다. 그 예제로써 프로세스 쌍 4(P4′)에서 고장이 발생한 그림 10을 본다. 쌍 2와 쌍 0,1의 경우 서로 메시지 이벤트를 완료하지만, 쌍 2,3,5 의 경우 고장이 발생한 P4′ 때문에 메시지 이벤트가 완료되지 못한다. 그룹핑 되어 있는 부분은 완료되지 못한 메시지 이벤트 때문에 블록 되어 있는 상태를 나타내고 있다. 블록 되어 있는 부분은 아래와 같은 과정을 통하여 일관성 조건을 만족시킨다.

1. 생성 알림 : 새로운 백업 프로세스의 생성을 상대방 프로세스 쌍에게 알린다.
2. 프로세스 상태 전송 : 생성된 후의 프로세스 상태를 알린다.
3. 상태 조정 과정 : 프로세스의 상태 정보를 이용해서 메시지 이벤트를 갱신한다.
4. 재시작 메시지 전송 : 복구 완료를 알린다.

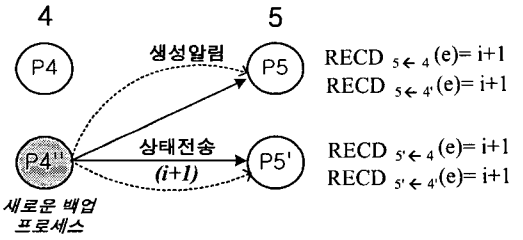


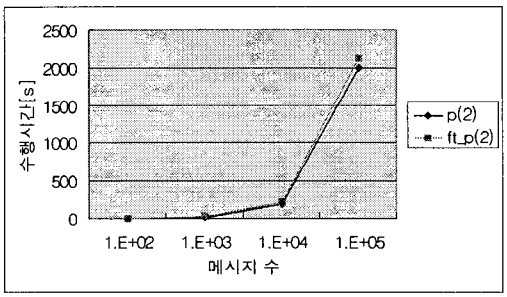
그림 11 프로세스간 일관성 유지 과정

프로세스 P4와 직접적인 메시지 교환이 없는 프로세스들 즉, P0, P1은 복구 후에 포트의 정보가 갱신되는 것 이외에는 아무런 영향도 받지 않기 때문에 복구 후 별다른 작업이 필요 없다. 그러나, 직접적인 메시지 교환이 있는 프로세스 P2, P3, P5는 새로 복구되는 프로세스의 상태가 변경되면서 일관성이 깨지게 된다. 그렇기 때문에 새로 생성되는 백업 프로세스는 복구 된 후 자신의 상태를 다른 프로세스들에게 알려 조정작업을 수행하게 된다. 예를 들어, 프로세스 P3과 P5에서의 일관성 유지과정을 보면 그림 11과 같다. 복구 과정에서 새로 생성된 프로세스 P4′′는 P5, P5′ 각각에 대하여 메시지 이벤트 정보를 보내서 서로의 상태를 일치시키게 된다. P5와 P5′프로세스는 P4′′ 프로세스로부터의 메시지 수신 이벤트를 i+1로 갱신하고 정리 1의 일관성

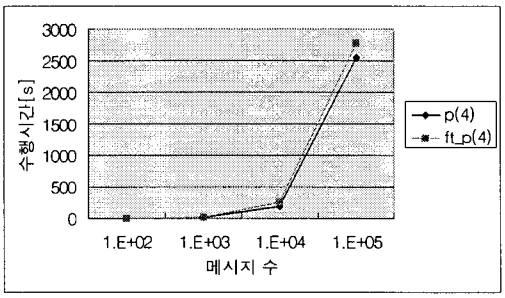
조건을 만족하게 된다. 그리고 다른 블록되어 있는 프로세스 P2와 P3 사이에도 위와 같은 동기 과정을 통하여 프로세스 P4와 일관성 문제를 해결한다.

4. 실험환경 및 결과

MPI는 메시지 전달을 위한 API(Application Programming Interface) 수준의 표준을 정의한 것으로 실제로 MPI를 구현하는 것은 시스템과 구현 의도에 따라 여러 가지 차이점을 가지고 있다. 또한 초기에 여러 노드들에 분산되어 있는 프로세스들을 실행시키는 방법과 프로세스들의 통신 방법 등에 많은 차이를 가질 수 있다. 본 논문에서는 공개된 MPI 구현 중에서 아르곤 국립 연구소(Argonne National Lab.)에서 개발한 MPICH를[4,11] 사용하였다. 실험에는 단일 노드의 Sun Ultrasparc 웍스테이션을 사용하였고, P4를[12] 이용하여 구현한 MPICH를 사용하였다. 실험은 FT-MPI의 성능 측정과 고장 허용 기능의 두 부분에 대하여 수행하였다.



(a)



(b)

그림 12 MPI, FT-MPI 프로세스 수행 시간 측정 (a) 프로세스 수가 2개일 경우 (b) 프로세스 수가 4개일 경우

4.1 FT-MPI 성능 측정

이전 MPI와 FT-MPI의 성능 비교를 위하여, 프로세스의 수행 시간을 비교하였다. 실험은 단일 노드의 Sun Enterprise 3000 웍스테이션을 사용하였다. 실험에 사용된 병렬 응용 프로그램은 덧셈을 10^2개 ~ 10^5개의 메시지 수만큼 반복 수행하는 프로그램으로, MPI 프로세스 수가 2개, 4개일 경우의 수행시간과 각각의 고장 허용 버전에서의 수행시간을 비교하였다.

그림 12는 각각의 경우에 대하여 메시지 수를 증가시켜 가면서 수행 시간을 측정한 것이다.

그림 12(a)에서 p(2)는 프로그램이 MPI 프로세스 두 개로 구성된 것을 말하고, ft_p(2)는 이것의 고장 허용 버전을 의미한다. ft_p(2)와 ft_p(4)의 경우 고장 허용을 위하여 p(2)와 p(4)에 비하여 두 배의 프로세스가 생성되어 수행된다. 그림 12에서 알 수 있듯이, 메시지 수를 최대 10^5개까지 실험한 결과에서 고장 허용 기능을 추가함으로써 수행 시간상에 미치는 오버헤드는 5% 범위를 넘지 않는다.

4.2 고장 허용 실험

실험은 단일의 Sun Enterprise 150 웍스테이션에서 수행하였으며, 고장 실험은 응용 프로그램의 프로세스들이 서로 일련의 메시지를 주고 받는 과정에서 한 프로세스를 임의로 종료하였을 때 고장 탐지 및 복구가 정상적으로 수행되는 지를 확인하는 것이다. 실험에 사용한 프로그램은 각각의 프로세스들이 1에서 6까지의 메시지를 전송하며 최종적으로 자신이 수신한 메시지의 합을 출력하는 프로그램이다.

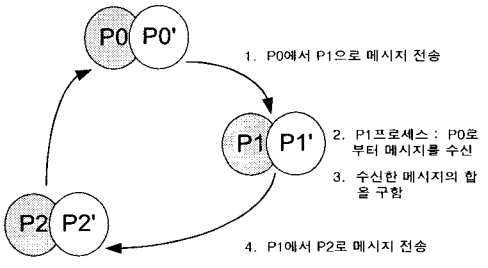


그림 13 고장 실험에 사용한 응용프로그램

테스트 프로그램의 고장 실험은 수행 도중에 구성 중인 어느 한 프로세스를 kill 명령어를 사용해 강제 종료시키는 방법을 사용하였으며 고장 허용 기능의 확인과 복구 여부, 최종 결과 값을 확인하였다. 실험에서는

세 쌍의 프로세스들이 사용되었고, 식별자 0은 식별자 3과 식별자 1은 식별자 4와 그리고 식별자 2는 식별자 5와 각각 프로세스 쌍 관계를 유지한다. 실험에서 식별자 값이 3인 프로세스를 수행 중 강제 종료 시켰다. 고장 발생한 프로세스는 자신의 쌍 프로세스에 의해서 새로운 백업 프로세스가 생성되고, 복구 과정을 통해서 고장 전과 마찬가지로 수행을 하게된다. 그림 14는 테스트 프로그램의 수행 결과를 보인 것으로서, 식별자 3프로세스는 고장 전 8022의 PID를 갖으며 SUM=3 이후 고장이 발생된다. 그 이후 결과 메시지들에서 PID가 8049인 식별자 3프로세스를 볼 수 있으며, 또한 프로세스들의 올바른 수행 결과를 통해서 고장 복구가 성공적으로 이루어졌음을 알 수 있다.

```
            ID  PID
             ↓  ↓
[ Real-ID : 3 (8022) ] SUM = 3  (고장 전)
                          ←(PID 8022 고장발생)
    .....

[ Real-ID : 2 (8012) ] SUM = 6
[ Real-ID : 0 (7994) ] SUM = 6
[ Real-ID : 5 (8040) ] SUM = 6
[ Real-ID : 0 (7994) ] SUM = 10
[ Real-ID : 1 (8003) ] SUM = 6
[ Real-ID : 4 (8031) ] SUM = 6
[ Real-ID : 5 (8040) ] SUM = 10
[ Real-ID : 2 (8012) ] SUM = 10
[ Real-ID : 1 (8003) ] SUM = 10
[ Real-ID : 4 (8031) ] SUM = 10
[ Real-ID : 3 (8049) ] SUM = 10 (고장 후)
[ Real-ID : 2 (8012) ] SUM = 15
[ Real-ID : 5 (8040) ] SUM = 15
[ Real-ID : 3 (8049) ] SUM = 15 (고장 후)
[ Real-ID : 1 (8003) ] SUM = 15
[ Real-ID : 4 (8031) ] SUM = 15
[ Real-ID : 0 (7994) ] SUM = 15
    .....
```

그림 14 실험 결과 메시지 요약

5. 결론 및 향후 연구

본 논문에서는 병렬컴퓨터 상에서의 메시지 전달 표준인 MPI에 고장 관리자 모듈을 추가함으로써 고장 허용 기능을 가지는 FT-MPI를 제안하였다. 제안한 FT-MPI는 추가적인 하드웨어 요구 없이 기존의 병렬 컴퓨터 상에서 수행이 되며, 또한 MPI와 완전히 호환되므로 기존의 MPI 프로그램들이 수정 없이 수행이 된다는 장점을 지니고 있다. 또한 고장 허용을 위해서 기존의 많이 사용하던 체크포인팅 방법을 벗어나, hot-spare의 프로세스 이중화 방법으로 고장 허용 기법을 제안하였다. 고장 실험은 Solaris2.x 운영체제를 탑재한 Sun사의 단일 웍스테이션 환경에서 수행되었으며, 구성중인 프로세스를 강제 종료시키는 고장 실험을 통하여 고장 허용 기능과 수행 결과를 확인하였다.

제안한 MPI 상에서의 고장 허용 방법은 고장 복구 등의 기능과 함께 프로세스간의 동기와 일관성 문제가 중요하다. 현재 프로세스간의 동기를 위한 문제 때문에 성능에 감소가 있으며, 다른 프로세서로의 프로세스 fork(즉, remote fork)와 같은 기능이 운영체제에서 제공되어야 한다는 문제점도 있다. 향후 연구 과제로는 위에서 언급한 문제점의 해결과, 고장 복구 도중에 또 다른 고장이 발생하는 경우와 같이 동시 다발적으로 발생하였을 때 이를 원할 히 처리할 수 있도록 고장 허용 기능을 보완하는 것이다. 또한 네트워 채널 상의 특성을 고려하여 일반적인 분산 시스템으로의 확장이 가능하도록 추가 연구가 필요하다.

참 고 문 헌

[1] Lou Baker, Bradley J. Smith, *Parallel Programming*, N.Y.:McGraw-Hill, 1997.

[2] Daniel P. Siewiorek, "Architecture of Fault-Tolerant Computers: An Historical Perspective," *Proc. IEEE*, Vol. 79, No. 12, pp. 1710-1734, Dec. 1991.

[3] Dhiraj K. Pradhan, *Fault-Tolerant Computer System Design*, Prentice Hall, 1996.

[4] MPI Home Page, http://www.mcs.anl.gov/mpi

[5] MPI Forum Home Page, http://www.mpi-forum.org

[6] PVM Home Page, http://www.epm.ornl.gov/pvm

[7] Flavin Cristian, "Understanding Fault-Tolerant Distributed Systems," *CACM*, Vol. 34, No. 2, pp. 56-78, Feb. 1991.

[8] D. B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing," *J. of Algorithms*, Vol. 11, pp. 462-491, 1990.

[9] Pankaj Jalote, *Fault Tolerance in Distributed Systems*,Englewood Cliffs, NJ:Prentice-Hall, 1994

[10] Richard Koo and Sam Toueg, "Checkpointing and Rollback -Recovery for Distributed Systems," *IEEE Trans. Software Engineering*, Vol. SE-13, No. 1, pp. 23-31, Jan. 1987.

[11] William Group and Ewing Lusk, "User's Guide for mpich, a Portable Implementation of MPI," MCSD. ANL., 1996.

[12] Ralph Butler and Ewing Lusk, "User's Guide to the

P4 Parallel Programming System," MCDS. ANL., April 1994.
[13] S. Venkatesan, T.-Y Juang, and A. Alagar, "Optimistic Crash Recovery without Changing Application Messages," *IEEE Trans. Parallel and Distributed Systems,* Vol. 8, No. 3, March 1997.
[14] Dhiraj K. Pradhan, "Roll-Forward Checkpointing Scheme: A Novel Fault-Tolerant Architecture," *IEEE Trans. Computers*, Vol. 43, No. 43, Oct. 1994.
[15] David E. Bakken, "Supporting Fault-Tolerant Parallel Programming in Linda," *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 3, March 1995.
[16] Walter H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," *ACM Computing Surveys*, Vol 13, No. 2, June 1981.

송 대 기
1997년 충남대학교 컴퓨터공학과(학사). 1999년 충남대학교 컴퓨터공학과(석사). 1999년 ~ 현재 충남대학교 컴퓨터공학과 박사과정. 관심분야는 운영 체제, 병렬 처리, 결함 허용, 실시간 시스템, Embedded System 등임.

이 철 훈
1979년 ~ 1983년 서울대학교 전자공학과 학사. 1983년 1986년 삼성전자 컴퓨터개발실 연구원. 1986년 ~ 1988년 한국과학기술원 전기및전자공학과 석사. 1988년 ~ 1992년 한국과학기술원 전기및전자공학과 박사. 1992년 ~ 1994년 삼성전자 컴퓨터사업부 선임연구원. 1994년 ~ 1995년 Univ. of Michigan 객원연구원. 1995년 2월 ~ 현재 충남대학교 컴퓨터공학과 조교수. 관심분야는 운영체제, 병렬처리, 결함 허용 및 실시간 시스템임.