상황 중심 프로그래밍을 기법을 이용한 결함 내성 MPI 시스템

한혁*, 정형수*, 김신규*, 염헌영*
*서울대학교 컴퓨터공학과
e-mail:{hhyuck, jhs, sgkim, yeom}@dcslab.snu.ac.kr

Fault-Tolerant MPI based on the Aspect-Oriented Programming

Hyuck Han*, Hyungsoo Jung*, Shin Gyu Kim*, Heon Y. Yeom*
*Dept of Computer Science and Engineering, Seoul National University

요 약

최근 상황 중심 프로그래밍(Aspect-Oriented Programming)에 관한 연구가 활발해져서 분산 및 병렬 시스템의 설계를 더 효과적으로 할 수 있게 되었다. 이 논문에서는 상황 중심 프로그래밍을 활용하여 분산 시스템의 전통적인 이슈 중의 하나인 결함 내성 시스템을 구축해보고자 한다.

1. 서론

최근 상황 중심 프로그래밍[1]에 대한 관심이 매우 커져가고 있다. 일반 객체 지향 프로그래밍의 단점을 극복하기 위해 등장했던 상황 중심 프로그래밍은 객체 지향 프로그래밍 및 절차적 프로그래밍에까지 적용되기 시작했다. 분산 시스템 특히 병렬 시스템의 전통적인 이슈 중에서 가장 중요한 것은 결함 내성 시스템에 관한 것이다. 이 논문에서는 상황 중심 프로그래밍을 이용하여 결함 내성 MPI 시스템을 구축하고, 이를 통해 상황 중심 언어가 분산 시스템 및 병렬 시스템에서 매우 효과적인 방법임을 보이고자 한다.

이를 위하여, 기존의 잡 관리 시스템, 검사점/복구 유틸리티[2], 미리넷 네트워크를 기반으로 하는 MPI (MPICH-GM)[3]을 이용한다. 즉, 상황 중심 언어를 기반으로 일관성있는 분산 검사점 및 복구 규약을 설계 및 구현하는 것이다. 또한, 실험 결과는 상황 중심 언어가 분산 시스템에서 널리 활용될 수 있음을 보여준다.

2. 상황 중심 언어

최근 객체 지향 프로그래밍은 절차적 프로그래밍을 완벽히 대체하였다. 하지만, 객체 지향 프로그래밍을 사용하더라도 코드 중복, 생산성 저하, 재활용성 저하 등의 문제를 피할 수 없다. 이를 극복하기 위해 나온 프로그래밍 방법이 상황 중심 언어이고, 아래의 4개의 개념을 기반으로 하고 있다.

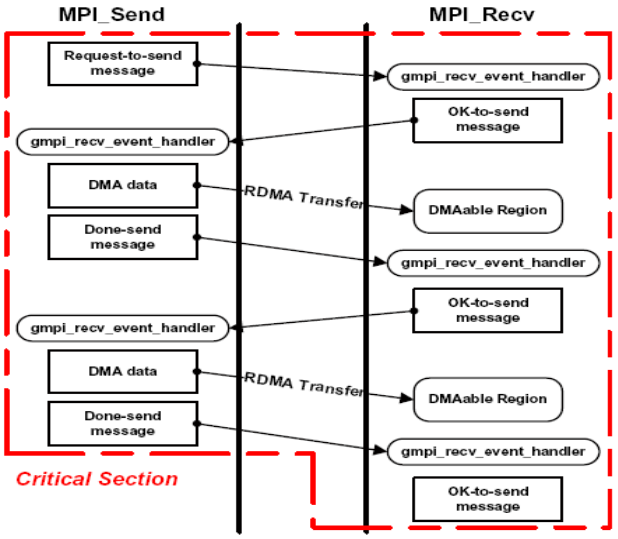
- 조인포인트(joinpoint) : 횡단 관심 모듈의 기능이 삽입되어 동작할 수 있는 실행 가능한 특정위치
- 포인트컷(pointcut) : 어떤 클래스의 어느 조인포인트를 사용할 것인지를 결정하는 선택
- 어드바이스(advice) : 각 조인포인트에 삽입되어져 동작할 수 있는 코드
- 위빙(weaving) : 포인트컷에 의해서 결정된 조인포인트에 지정된 어드바이스를 삽입하는 과정

이런 4개의 개념을 이용하여 기존의 코드를 수정하지 않고 새로운 기능을 첨가할 수 있다.

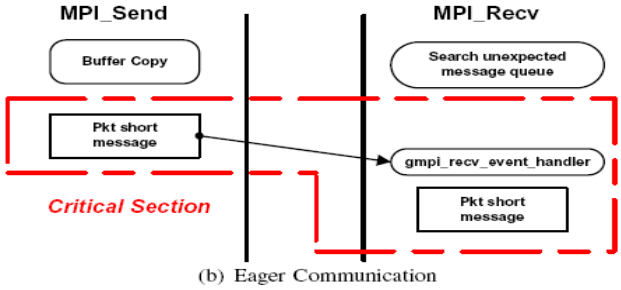
3. 결함 내성 MPI 시스템

초기화. 초기화 부분은 MPICH-GM이 잡 관리 시스템과 연동되는 부분이다. 원래 MPICH-GM의 초기화 부분을 상황 중심 언어를 사용하여 MPID_VMI_Init 함수를 MPID_VMI_Initialize_via_JMS로 바꾼다.

일관적인 분산 검사점 규약. 일관적인 분산 검사점 규약은 결함 내성 MPI의 핵심 부분이 된다. 이를 위해 먼저 MPICH-GM의 송수신 함수를 분석하여 절대로 검사점이 잡혀서는 안 될 영역을 정의한다. 그림 1은 MPICH-GM의 보호 영역을 보여준다. 검사점 신호가 각 MPI 프로세스에 전달되면 MPI 프로세스는 해당 프로세스가 보호 영역을 수행하지 않고, 보내고 있거나 받고 있는 메시지가 없는 경우에 검사점 가능 상태로 진입하게 된다. 검사점 가능 상태가 되면, 개별 프로세스는 참여하고 있는 모든 프로세스에 스냅샷 메시지를 브로드 캐스팅한다. 이는 [4]에서 제시한 분산 스냅샷 알고리즘과 비슷하다. 브로드 캐스팅 된 메시지를 주고받는 것은 현재 MPI 프로세스가 송신 함수를 끝냈지만, 반대쪽 프로세스가 아직 수신하지 못 한 메시지(in-transit message)를 MPICH-GM의 unexpected queue에 저장하게 해준다. 그리고 차후에



(a) Rendezvous Communication



(b) Eager Communication

그림 1 보호 영역

그곳에서 필요한 메시지를 얻어온다. 브로드 캐스팅이 끝나게 되면 개별 MPI 프로세스는 두 가지 중요한 조건을 보장받게 된다. **"전송중인 메시지(in-transit message)와 고아 메시지(orphan message)가 없다"**는 것을 확실히 보장받게 된다. 이후에 개별 프로세스는 검사점을 안전한 저장 장치에 남긴다. 따라서 상황 중심 언어를 사용하여 보호 영역을 구현하고, 검사점을 호출해야 한다. 그림 2는 Eager 규약에 대한 보호 구역 구현의 예이다.

```
ReturnType around CEss () on (Jp): call(Jp,"MPID_CH_Eagerb_send_short")
&& type(Jp, ReturnType)
{
  ReturnType i ;
  crtical_section++;
  i = proceed ();
  crtical_section--;
  if( critical_section == 0 && ckpt_signal_received ) do_ckpt();
  return i;
}
```

그림 2 상황 중심 언어로 작성한 보호 구역(Eager 규약)

일관적인 분산 복구. 일관적인 분산 복구는 장애의 종류에 따라서 절차가 다르다. 프로세스 장애인 경우는 잡 관리 시스템의 지역 관리자가 최신 검사점을 기반으로 다시 개별 MPI 프로세스들을 동작시킨다. 노드 장애인 경우에는 최근 검사점을 안전한 원격 저장 장치에서 읽어온 후에 다른 노드에 지역 관리자를 동작시키고, 동작 관리자가 MPI 프로세스를 기동한다. MPI 프로세스를 다시 시작하게 되면, 기존의 병렬 작업에 다시 참여하여야 하고, 노드의 위상 및 위치를 재조정해야 한다.

```
ReturnType around ckptAdv () on (Jp): call(Jp,"ckpt_ckpt")
&& type(Jp, ReturnType)
{
  ReturnType i ;
  i = proceed (); /*ckpt_ckpt*/
  if ( ret != 0 )
  {
    //rejoin
    //device reconfiguration
    //topology reconfiguration
  }
}
```

그림 3 검사점 어드바이스(복구)

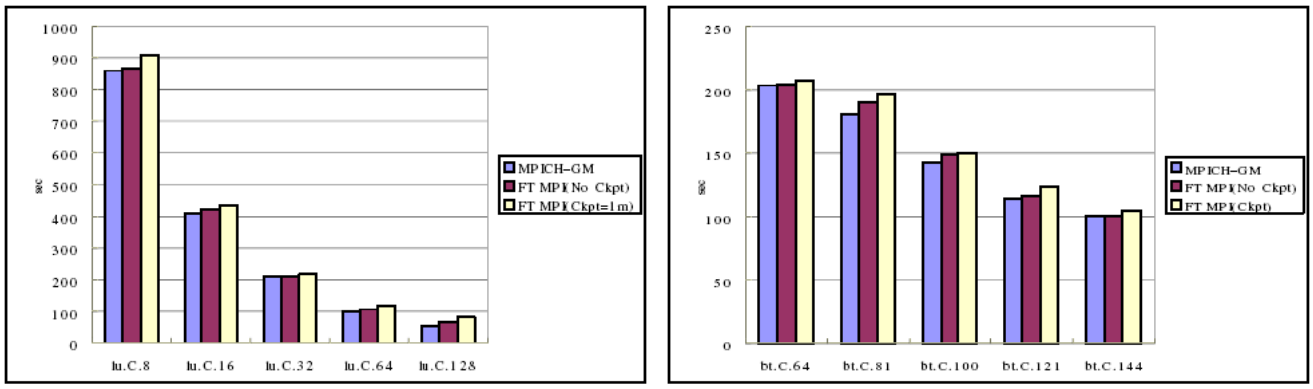
그림 3은 일관적인 분산 복구 절차를 상황 중심 프로그래밍 기법을 사용하여 작성한 것이다. rejoin, device reconfiguration, topology reconfiguration이 복구 절차를 나타내는 것이다.

앞에서 언급한 세 가지 부분을 상황 중심 프로그래밍 기법을 사용하여 결함 내성 MPI 시스템을 구축하는데 사용하였다. 상황 중심 언어는 개별적인 소프트웨어 컴포넌트와 그것들 사이의 관계(interaction)을 분리할 수 있어서 더욱 코드 관리가 쉽고, 코드를 읽기 편하게 유지할 수 있다. 또한 위의 방식은 MPICH-G2[5], MVAPICH[6]와 같은 다른 MPI 구현물에서도 쉽게 적용할 수 있다.

4. 실험

실험은 한국과학기술정보원에서 운영하고 있는 Hamel 클러스터 시스템에서 수행하였다. Hamel 클러스터의 개별 노드는 두 개의 Intel Xeon 2.8GHz CPU, 3GB의 메모리 그리고 30GB의 SCSI 디스크가 장착되어 있다. 실험을 위하여 NPB 벤치마크의 LU와 BT를 이용하였다. LU의 경우에는 1분에 한 번, BT의 경우에는 전체적으로 한 번 검사점을 남기게 하였다.

그림 4는 LU와 BT의 수행 시간을 보여준다. MPICH-GM은 원래의 미리넷 기반 MPI 구현물을 사용했을 때의 결과이고, FT-MPI(No Ckpt)는 구축된 시스템으로 수행하되, 검사점을 남기는 않는 방법으로 실험했을 때의 결과이다. LU, BT의 경우 모두 수행 시간의 차이가 거의 나질 않는다. FT-MPI(Ckpt)의 경우에는 LU/BT 모두 수행 시간이 검사점을 남기지 않는 것 혹은 원래 MPICH-GM의 경우보다 느리게 되는데, 이것은 검사점 오버헤드이다. 이것은 피할 수 없는 오버헤드이지만, 그다지 크지 않고, [7]에서의 실험 결과 양상과 같다. 이것은



(a) LU (b) BT

그림 4 수행 시간

상황 중심을 사용해서 구현하는 것과 사용하지 않고 구현하는 것 사이의 성능의 차이가 없다는 것을 의미한다.

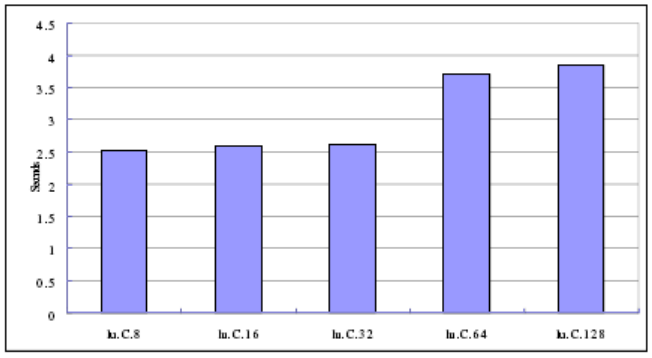


그림 5 복구 시간

그림 5는 LU의 복구 시간을 보여준다. 복구 시간 역시 [7]에서의 결과가 매우 유사하다. 따라서 상황 중심 프로그래밍을 사용하여 결함 내성 MPI 시스템을 구현하는 것과 그것을 사용하지 않고 결함 내성 MPI 시스템을 구현하는 것과 성능 차이가 거의 없고, 오히려 코드의 가독성 및 코드의 유지 관리 측면에서 유리한 점이 있음 알 수 있다.

5. 결론

이 논문은 상황 중심 프로그래밍을 사용하여 분산 시스템의 오랜 이슈 중의 하나인 결함 내성 MPI를 구현하고, 실험한 결과를 보여준다. 실험 결과를 통하여 우리의 접근 방법이 성능 상의 문제가 없으며 오히려 코드 관리 측면에서 장점이 있음을 알 수 있다. 또한, 이런 방법이 다른 MPI 시스템에도 쉽게 적용됨을 알 수 있다.

감사의 글

이 연구는 서울시 산학연 협력 사업에서 부분적으로 지원 받았으며, 서울대학교 컴퓨터 연구소는 이 연구의 시설을 제공하였습니다.

참고문헌

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, “Aspect-Oriented Programming." In Proceedings of European Conference on Object-Oriented Programming, 1997.

[2] V. C. Zandy, ckpt. http://www.cs.wisc.edu/zandy/ckpt.

[3] Myricom, Myricom Homepage. http://www.myri.com.

[4] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems." ACM Transactions on Computer Systems, vol. 3, no. 1, 1985.

[5] N. T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: a Grid-enabled implementation of the Message Passing Interface," Journal of Parallel and Distributed Computing, 2003.

[6] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High Performance RDMA-Based MPI Implementation over InfiniBand," in Proceedings of 17th Annual ACM International Conference on Supercomputing, 2003.

[7] H. Jung, D. Shin, H. Han, J. Kim, H. Yeom, and J. Lee, "Design and Implementation of Multiple Fault-Tolerant MPI over Myrinet," in Proceedings of SC'05, 2005.