자바 메시지 전달 시스템에서의 결함 포용 병렬 애플리케이션*

안 진 호, 김 기 범, 김 정 훈, 황 종 선
고려대학교 컴퓨터학과 분산시스템 연구실

Fault-Tolerant Parallel Applications in Java Message Passing Systems

Jin Ho Ahn, Kibom Kim, Jeong Hoon Kim, Chong Sun Hwang
Distributed Systems Lab., Dept. of Computer Science, Korea University

요약

동기적 검사점(synchronous checkpointing) 기법, 인과적 메시지 로깅(causal message logging)과 향상된 회복 비동기성(improved asynchronism during recovery)을 제공하는 복귀회복(rollback recovery) 기법을 적용하여 자바 메시지 전달 시스템(java message passing system)에서 수행하는 병렬 애플리케이션들에게 저 비용의 결함 포용성(fault-tolerance)을 제공하는 소프트웨어 패키지를 구현하고자 한다. 최근에 통신과 프로세서 기술이 급속도로 발전함에 따라, 통신망으로 연결된 이질형(heterogeneous) 컴퓨터들을 이용하는 대규모 분산 시스템들은 아주 효율적인 병렬 컴퓨팅 환경을 제공해준다. 그러나, 이러한 분산 시스템들의 규모가 커짐에 따라 고장률(failure rate)도 그만큼 증가하게 된다. 따라서, 고장률이 높은 대규모 분산 시스템들에게 좀 더 효율적인 결함 포용성을 제공하는 기법들이 필요하다. 또한, 대규모 분산시스템들은 이질형 컴퓨터들로 구성되어 있기 때문에, 결함 포용성을 제공하는 소프트웨어 패키지들은 플랫폼 독립적(platform independent)이어야 한다. 이러한 문제점은 높은 이식성(portability)을 가지고 있는 자바 언어로 구현함으로써 해결될 수 있다.

따라서, 본 논문은 자바 메시지 전달 시스템에서 수행하는 병렬 애플리케이션들에게 동기적 검사점 기법, 인과적 메시지 로깅과 향상된 회복 비동기성을 제공하는 복귀회복 기법을 높은 이식성을 가진 자바 언어로 구현하여 저 비용으로 결함 포용성을 제공하고자 한다.

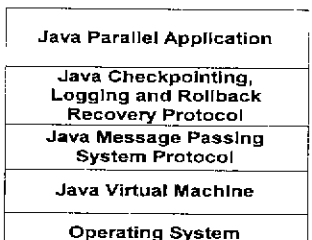
1. 서론

최근에 통신과 프로세서 기술이 급속도로 발전함에 따라, 통신망으로 연결된 이질형 컴퓨터들을 이용하는 대규모 분산 시스템들은 아주 효율적인 병렬 컴퓨팅 환경을 제공해준다. 따라서, 이러한 병렬 컴퓨팅 환경에서 장시간 수행하는 병렬 애플리케이션들이 통신망으로 연결된 각 컴퓨터기 가지고 있는 가용한 모든 자원들을 동시적으로 이용할 수 있다 그러나, 첫 번째 문제점으로 분산 시스템들의 규모가 커짐에 따라 고장률도 그만큼 증가하게 된다 따라서, 고장률이 높은 대규모 분산 시스템들은 좀 더 효율적인 결함 포용성을 제공하는 기법들을 필요로 한다. 두 번째 문제점으로 대규모 분산시스템들은 이질형 컴퓨터들로 구성되어 있기 때문에, 메시지 전달 서비스들과 결함 포용성을 제공하는 소프트웨어 패키지들은 플랫폼 독립적이어야 한다

따라서, 본 논문에서 첫 번째 문제점은 동기적 검사점[5], 인과적 메시지 로깅[2]과 향상된 회복 비동기성을 제공하는 복귀회복 기법[1]을 제공하는 소프트웨어 패키지를 구현함으로써 해결된다 자바는 높은 코드 이식성과 통신에 관련하여 운영체제에 독립적으로 동일한 인터페이스를 제공한다 따라서, 두 번째 문제점은 자바 메시지 전달 시스템에 기반하여 결함 포용성을 제공하는 소프트웨어 패키지를 자바로 구현함으로써 해결된다.

2. 관련연구

통신망으로 연결된 이질형 컴퓨터들을 이용하는 병렬 컴퓨팅 환경은 일반적으로 메시지 전달 모델을 제공하는 소프트웨어 패키지들에 기반하고 있다. 이러한 메시지 전달 모델들 중 가장 널리 사용되는 것들로는 Parallel Virtual Machine(PVM)[7]과 Message Passing Interface(MPI)[6]가 있다 이러한 메시지 전달 모델들은 분산 시스템들의 규모가 커짐에 따라 증가하는 고장률에 대한 문제의 해결책을 포함하고 있지 않다. 따라서, PVM이나 MPI과 같은 메시지 전달 모델에는 결함 포용성을 제공하는 기법들을 필요로 하게 된다 이러한 문제점을 해결하기 위한 기존 연구들로는 대표적으로 PVM 환경에서 구현된 것[3]과 MPI 환경에서 구현된 소프트웨어 패키지[8]가 있다. 그러나 이러한 모든 패키지들은 대규모 이질형 분산시스템들의 플랫폼 독립성에 대한 문제점을 해결하지 못한다 예를 들면, PVM과 MPI 환경에서 구현된 패키지들은 프로세스의 검사점을 생성함에 있어서 몇가지의 운영체제-의존적인 부분들이 포함되게 된다 이러한 경우, A 컴퓨터상에서 수행하던 한 프로세스가 하나의 검사점을 생성시키고 이후에 고장이 발생하여 B 컴퓨터상에서 회복하여 재시작한다고 하자 이때, A와 B의 운영체제가 다르면 A 컴퓨터에서 생성된 프로세스의 검사점을 B 컴퓨터에서 사용할 수 없다. 따라서, 본 논문에서는 이러한 대규모 이질형 분산시스템들의 플랫폼 독립성에 대한 문제점을 해결하는 소프트웨어 패키지를 구현하고자 한다



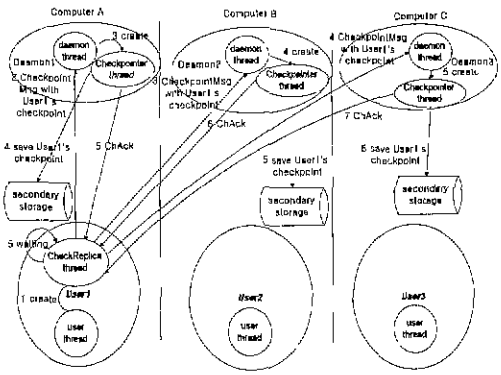
<그림 1> 전체 시스템 계층도

3. 구현

3.1 전체 시스템 구성도

본 논문에서 구현된 소프트웨어 시스템의 계층도는 그

*이 연구는 한국과학재단 '98 핵심전문연구과제(981-0924-126-2)인 "분산시스템에서 비동기적 회복기법의 개발"의 지원을 받은 것임



<그림 2> User1 프로세스의 검사점을 저장하는 예

림 1과 같다. 각 이질형 컴퓨터상에는 운영체제 계층과 그 상위 계층에 자바 가상 머신이 있다 그리고, 그 위에 통신망으로 연결된 이질형 컴퓨터들을 하나의 가상 병렬 컴퓨팅 환경으로 만들어주는 자바 메시지 전달 시스템 프로토콜 계층이 놓이게 되고, 그 상위 계층에는 동기적 검사점 기법, 인과적 메시지 로깅과 향상된 회복 비동기성을 제공하는 복귀회복 프로토콜이 놓이게 된다 최상위계층에는 수행하고자 하는 자바 병렬 애플리케이션이 있다

3.2 메시지 전달 시스템

본 논문에서는 이질형 컴퓨터들간의 메시지 전달을 위해 자바 메시지 전달 시스템(JMPS)을 사용하고 있다. JMPS는 JPVM[4]과 같이 순수하게 자바에서 제공하는 패키지들만 이용하여 구현된 것으로 PVM메시지 전달 모델을 따르고 있다. 즉, 병렬 컴퓨팅 환경을 구성하고 있는 각각의 이질형 컴퓨터들에는 하나의 자바 병렬 애플리케이션의 수행을 위한 프로세스들의 생성과 관리를 담당하는 하나의 JMPSDaemon이 존재한다 그리고, 모든 JMPSDaemon들에 대한 정보 제공 및 추가, 삭제와 프로세스들의 수행 상태 정보를 제공하는 하나의 JMPSController가 있다. 또한, JMPS는 각 프로세스마다 하나의 메시지 큐를 가지도록 구현되었기 때문에 프로세스들간에 메시지들을 FIFO형태로 전달해준다.

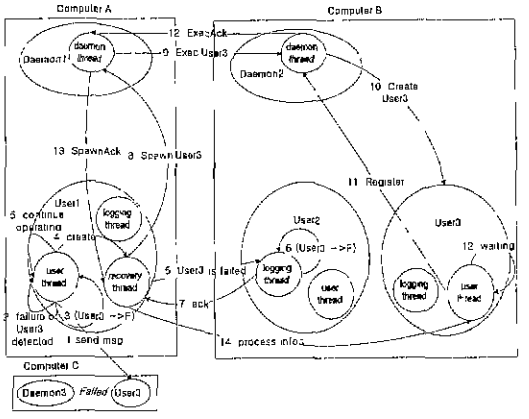
3.3 검사점과 메시지 로깅 기법

3.3.1 동기적 검사점 기법

본 논문에서 사용하고 있는 검사점 기법은 동기적 검사점 기법이다[5] 즉, 하나의 자바 병렬 애플리케이션의 수행을 시작한 프로세스가 검사점 조정자(checkpointing coordinator)가 되어 주기적으로 모든 프로세스들에게 동기적으로 검사점을 취하게 하여 프로세스들간의 일관된 전역 상태(consistant globally state)[1]를 유지하도록 한다. 또한 각 프로세스는 검사점을 지장함에 있어서 그림 2와 같이 병렬 컴퓨팅 환경을 구성하는 가용한 이질형 컴퓨터들의 지역 디스크상에 자신의 검사점을 JMPSDaemon을 통해서 복사시킨다. 이러한 방법은 네트워크상에 연결된 중앙 집중형 대규모 RAID 디스크와 같이 안정된 저장소(stable storage)를 구현하는 방법의 문제점들인 단일고장(single point failure)과 병목 현상에 따른 성능저하(performance bottleneck)를 없애준다. 또한 각 컴퓨터에 한 프로세스의 검사점을 저장하는 동안에 그 컴퓨터에 시스템 고장(system failure)이 발생하더라도 일관된 검사점으로 회복할 수 있도록 [9]에서 제안한 방법을 사용했다 즉, 이것은 한 프로세스의 검사점을 위해 하나의 헤더파일과 두 버전의 검사점 파일들을 사용한다 헤더파일에는 하나의 boolean형 데이터로 구성되는데 한 프로세스의 검사점을 저장하기 전에 이 데이터 값이 'true'이면 첫 번째 버전 파일에 저장하고 이 데이터 값을 'true'에서 'false'로 변경한다. 또한 한 프로세스의 검사점을 저장하기 전에 이 데이터 값이 'false'면 두 번째 버전 파일에 저장하고 이 데이터 값을 'false'에서 'true'로 바꾼다.

3.3.2 인과적 메시지 로깅 기법

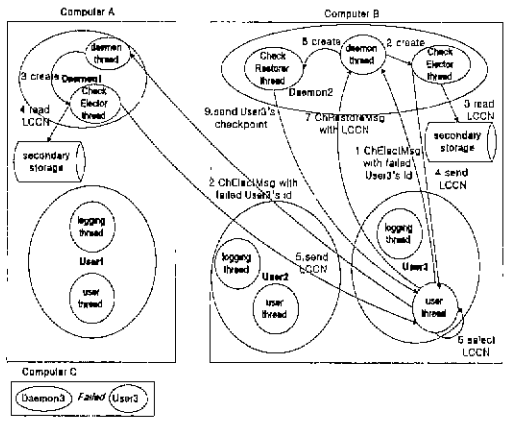
본 논문에서 구현된 소프트웨어 패키지의 정상수행시 사용되는 메시지 로깅 기법은 인과적 메시지 로깅 기법[2]에 기반한다. 이 기법에서 임의의 메시지 m은 다음과 같은 5가지 속성들을 가진다.*m.source*는 *m*의 송신 프로세스, *m.dest*는 *m*의 수신 프로세스, *m.data*는 *m.source*가 *m.dest*에게 전달하려는 데이터, *m.ssn*은 *m.source*에서의 *m*에 대한 송신번호 그리고 *m.rsn*은 *m.dest*에서의 *m*에 대한 수신번호이다 이 기법은 *f*개의 프로세스들의 동시적 고장을 포용하기 위해서는 메시지 송신자의 휘발성 저장소에 그 메시지의 속성인 (*source, dest, data, ssn*)정보를 저장하고, *f*+1개의 다른 프로세스들의 휘발성 저장소에 그 메시지의 결정자(determinant) 즉, (*source, dest, ssn, rsn*)정보를 저장해놓으면 충분하다는 것이다. 이 기법의 장점들은 시스템을 구성하는 프로세스들의 수가 *n*이라고 할 때 *f*개의($1 \le f < n$) 동시적인 고장들을 포용하기 위해서 안정된 저장소에 이러한 로깅정보를 동기적으로 저장할 필요가 없고, 정상수행시 프로세스의 수행을 방해하지 않기 때문에 이 기법에 의한 성능에 미치는 영향이 적다는 것이다. 부가적으로, 각 프로세스는 수행하고 있는 자바 병렬 애플리케이션에 포함된 모든 프로세스들의 재현번호(incarnation number)와 상태값을 저장하는 프로세스 테이블(ProcessTable)을 가지고 있다. 재현번호란 한 프로세스가 고장이 발생하여 회복한 횟수를 말한다. 각 프로세스는 하나의 메시지를 다른 프로세스에게 송신할 때 자신의 재현번호를 포함해서 보낸다. 그리고, 각 프로세스의 상태란 현재 살아있는 상태('Live') 인지, 고장난 상태('Failure')인지, 회복중인 상태('Recovering')인지를 나타내는 것이다 이들은 회복시 향상된 회복 비동기성을 제공하는 복귀회복 기법을 사용하기 위해 필요한 데이터 구조이다[1].



<그림 3> User3 프로세스 고장시 재생성시키는 예

3.4. 회복 기법

본 논문에서 사용하는 회복기법은 향상된 회복 비동기성을 제공하는 복귀회복 기법이다[1] 예를들면, 그림 3과 같이 C 컴퓨터가 고장이 발생한 경우, A 컴퓨터에 있는 프로세스 User1이 C 컴퓨터에서 수행하던 프로세스 User3에게 메시지를 전송하려할 때, User1은 User2가 고장이 났다는 사실을 알게된다 이 때, User1은 자신의 ProcessTable에서 User3의 상태를 'Fail'로 변경하고, User3를 다시 가용한 컴퓨터에 생성시키고 다른 프로세스들의 정보를 전달하는 책임을 가지는 recovery thread를 생성시킨다. 그리고, user thread는 자신의 컴퓨팅을 계속한다 User1의 recovery thread는 다른 프로세스들에게 User3가 고장이 발생했다는 사실을 전달해주고, 그들로부터 응답 메시지를 기다린



<그림 4> User3 프로세스가 최근의 검사점으로 회복하는 예

다 이들을 받은 후에 User1의 recovery thread는 A 컴퓨터에 있는 JMPSDaemon1에게 User3를 위한 프로세스를 생성시키도록 요청한다. 그러면 JMPSDaemon1이 B 컴퓨터에 있는 JMPSDaemon2에게 User3를 위한 프로세스를 생성시키도록 한다. 이때, JMPSDaemon2는 User3를 위한 새로운 프로세스를 생성시킨다. 그리고 나서, User1에 있는 recovery thread가 User3를 위한 프로세스가 생성되었다는 응답메시지를 JMPSDaemon2로부터 JMPSDaemon1을 통해 수신했을 때, 자신이 가지고 있는 프로세스들에 관한 정보들을 재생성된 User3 프로세스에게 송신해주고, recovery thread는 소멸된다. 그 후에 User3는 이전의 검사점을 얻게 되는데, 그 과정은 그림 4와 같다. 먼저, User3은 모든 JMPSDaemon들에게 자신의 검사점들중 가장 최근의 검사점 번호(LCCN)을 요청한다. 그리고, 각각의 JMPSDaemon으로부터 LCCN들을 받은 후에 이들중 가장 큰 값을 선택하여 이 LCCN의 검사점 파일을 저장하고 있는 JMPSDaemon에게 User3의 검사점을 전송하도록 요청한다 그러면, 그 JMPSDaemon은 CheckRestorer thread를 생성시켜서 이 검사점을 전송해준다. User3는 이 검사점까지 회복한 후, User1과 User2에게 현재 회복중이라는 정보와 각각의 상태 정보를 요청하는 메시지를 보낸다. 그러면, User1과 User2의 logging thread가 자신의 상태정보를 User3에게 전달해 준다. 이 때, User3는 모든 프로세스들의 상태 정보를 얻은 후에 이들 중 살아있는 프로세스들 즉, User1과 User2에게 자신이 회복하는데 필요한 결정자들을 요청하는 메시지를 전송한다. 그러면, User1과 User2의 logging thread에서 User3로부터 요청된 결정자들을 모아서 User3에게 전송해 준다. User3는 이러한 결정자들을 모두 모으고, 재연(replay)하게 된다 이러한 User3의 회복중에 User1과 User2의 user thread는 블록되지 않고 계속해서 수행한다

4. 실험 및 성능평가

Application	No checkpointing and logging	checkpointing and logging
PQuickSort	6,598 seconds	7,198 seconds

<표 1> 애플리케이션들의 수행시간 비교표

본 논문에서 구현된 소프트웨어의 실험은 32MB 메모리와 하나의 100MHZ 펜티엄 프로세서상에서 Windows 95 운영체제를 기반으로 하는 PC 3대와 6개의 UtraSPARK processor들과 512MB 메모리상에서 Solaris 2.6 운영체제를 기반으로 하는 Sun Enterprise 3000 1대를 10 Mbit/sec의 Ethernet으로 연결한 환경에서 수행되었다. 실험에 사용된 애플리케이션은 PQuickSort이다 이 애플리케이션은 2,000,000개의 정렬되지 않은 키(key)들을 정렬하는 병렬 퀵정렬(parallel quicksort)프로그램이다. PQuickSort를 수행하기 위해 5개의 프로세스들이 생성되었고, 전체적으로 수행하는 동안 20번의 검사점들을 취했다. 표 1에서 보는 바와 같이 PQuickSort가 결함포용성을 제공하지 않는 자바 메시지 전달시스템에서 수행하는데 걸린 시간과 결함포용성을 제공하기 위해 본 논문에서 구현된 소프트웨어 패키지에서 수행하는데 걸린 시간과의 차이는 600초이다. 즉, 총 수행시간의 8.3%정도가 결함포용성을 제공하기 위해 발생되는 비용이 된다 이 결과는 본 논문에서 구현된 소프트웨어 패키지가 결함 포용성을 제공함에 있어서 매우 적은 비용을 발생시킨다는 것을 보여준다

5. 결론

본 논문에서 언급된 결함 포용성을 제공하는 자바소프트웨어 패키지는 첫 번째로 동기적 검사점, 인과적 메시지 로깅과 향상된 회복 비동기성을 제공하는 복귀회복 기법을 적용함으로써, 분산 시스템들의 규모가 커짐에 따라 증가하는 고장률에 대한 문제점을 해결하고, 두 번째로 자바 메시지 전달 시스템에 기반하여 순수하게 자바로 구현됨으로써 대규모 이질형 분산시스템들의 플랫폼 독립성에 대한 문제점을 해결했다. 또한, 성능 평가에서 살펴본 바와 같이 본 논문에서 구현된 소프트웨어 패키지는 결함 포용성을 제공함에 있어서 매우 적은 비용을 발생시킨다는 것을 알게되었다.

[참고문헌]

[1] 안진호, 정광식, 박찬열, 김기범, 황종선, "인과적 메시지 로깅에 기반한 향상된 회복 비동기성을 제공하는 회복 기법," *한국정보과학회 춘계학술발표회 논문집(A), 25권 1호*, pp. 605-607, 1998.

[2] L. Alvisi and K Marzullo, "Message logging: Pessimistic, optimistic, causal, and optimal," *In Proceedings of the 15th International Conference on Distributed Computing Systems*, May 1995.

[3] J.Leon, A. L. Fisher, and P Steenkiste, "Fail-safe PVM: a portable package for distributed programming with transparent recovery," *Technical Report CMU-CS-93-124*, Carnegie Mellon University, February 1993

[4] Adam J Ferrari, "JPVM: Network Parallel Computing in Java," *Technical Report CS-97-29*, University of Virginia, December 1997

[5] R. Koo and S Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, 13(1):23-31, January 1987

[6] W Gropp, E. Lusk, and A Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, 1994

[7] A.Geist, A Beguelin, J. Dongarra, W Jiang, R Manchek, and V S. Sunderam, *PVM: Parallel Virtual Machine*, MIT Press, 1994.

[8] G. Stellner, "CoCheck: Checkpointing and process migration for MPI," *In Proceedings of International Parallel Processing Symposium, IEEE*, April 1996.

[9] F. Cristian, S Mishra, and Young S. Hyun, "Implementation and Performance of a Stable-Storage Service in Unix," *In Proceedings of the 15th Symposium on Reliable Distributed Systems, IEEE*, 1996.