분산환경에서 병렬프로그램 재실행을 위한 자바 디버거

최동순 김남훈 김명호
숭실대학교 컴퓨터학과
{dschoi, nhkim}@ss.soongsil.ac.kr, kmh@comp.soongsil.ac.kr

A Java Debugger for Replaying Parallel Programs on a Distributed Environment

Dong-Soon Choi Nam-Hoon Kim Myung-Ho Kim
Dept. of Computer Science, Soongsil University

요 약

네트워크 처리속도의 증가로 네트워크 컴퓨팅 환경에서의 작업이 늘고 있다. 플랫폼 독립성이라는 특성을 내세운 자바는 일반적인 이 기종간의 네트워크에서의 프로그래밍 언어로 많이 이용되고 있다. 그리고 이러한 네트워크 컴퓨팅 환경에서 병렬 프로그램 디버깅의 어려움으로 인해 자바 병렬 프로그램을 위한 디버거의 필요성이 요구되고 있다. 기존의 디버거들은 이런 병렬 환경에서의 디버깅을 각 프로세스에 하나의 순차디버거를 붙인 디버깅 환경을 제공한다. 그러나 병렬 프로그램은 순차프로그램과 다른 재실행시의 비결정적인 특성을 가지고 있음으로 일반적인 순차 디버거를 이용한 디버깅은 의미가 없다. 본 논문에서는 자바로 구현된 네트워크 컴퓨터(JaNeC)에서 병렬프로그램을 디버깅하기 위하여 재실행 시 실행 순서를 보장하는 자바 디버거를 소개한다.

1. 서 론

네트워크 처리속도의 증가로 네트워크 컴퓨팅 환경에서의 작업이 늘고 있다. 네트워크 컴퓨터는 네트워크상의 이기종 컴퓨터, 대용량의 기억장치, 과학 연산 장치 등 지리적으로 분산되어 있는 각 자원들에 접근하여, 이것들을 공간적으로 제약 받지 않고 사용할 수 있도록 사용자들에게 인터페이스를 제공하는 컴퓨팅 작업환경을 말한다 [1]. 이러한 네트워크 컴퓨팅 환경을 개발하는 개발자들이 제일 먼저 마주치는 문제는 서로 연결되어 있는 컴퓨터들이 다양하다는 것이다. 이러한 서로 다른 구조상의 환경에서 자바는 플랫폼 독립적인 특성으로 인하여, 이기종 분산 네트워크 환경에서의 프로그래밍 언어로 많이 이용되고 있다.

이와 같은 네트워크 컴퓨팅 환경을 제공하는 시스템으로 숭실대학교 시스템 소프트웨어 연구실에서 개발한 자바 기반 병렬 프로그래밍 환경인 JaNeC이 있다. JaNeC에서의 애플리케이션들은 자바로 작성된 프로그램으로 병렬 환경을 구성하는 여러 개의 호스트에서 병렬적으로 수행되는 프로그램들이다. JaNeC에서 사용하는 디버거로는 기존의 JDK안에 있는 원격 호스트에 대한 디버깅이 가능한 jdb라는 디버거를 사용한다. 이 jdb를 사용하여 프로그래머는 지역 또는 원격의 자바 가상머신에서 수행 중인 프로그램에 접근하여 클래스에 관한 많은 정보를 알아내고 디버깅할 수 있다. 하지만 jdb는 명령어 라인에서 디버깅을 하는 것으로 사용상의 불편함과 함께 재실행시 실행순서의 보장 등 병렬 프로그램 디버깅의 한계를 가지고 있다.

따라서 본 논문에서는 자바 병렬 프로그래밍환경인 JaNeC에 추가되어질 GUI를 기반으로 한 재실행 자바 병렬 디버거를 제시한다.

2. JaNeC에서의 병렬 자바 디버거의 설계

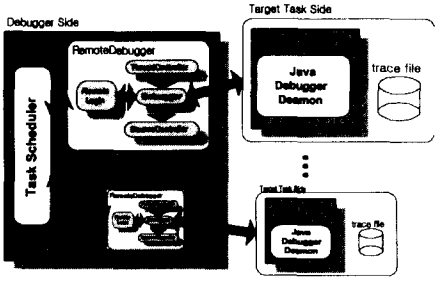

그림 1 디버거의 구성 요소

재실행 자바 디버거는 디버거가 수행되는 호스트의 컴포넌트와 태스크가 수행되는 호스트의 컴포넌트인 디버거 데몬, 그리고 트레이스 파일등 크게 세 부분으로 나눌 수 있다. 자바 디버거가 수행되는 호스트의 컴포넌트로는 원격 로그인 (Remote Login)과 쓰레드

컨트롤러 (ThreadController), 소스 컨트롤러 (SourceCont- roller), 쓰레드 스케줄러(Thread Scheduler)가 있고 목적 프로그램측의 컴포넌트로는 디버거 데몬(Debugger daemon)이 있다. 그리고 재실행을 위한 정보를 기록하고 있는 트레이스 파일(trace file)이 있다. 전체적인 구조는 그림 1과 같다.

2.1 쓰레드 컨트롤러

쓰레드 컨트롤러는 사용자의 프로그램이 수행되면서 발생되는 쓰레드들을 리스트에 나열시키고 쓰레드들을 중단(suspend)시키고 다시 동작(resume)하게 할 수 있는 간단한 오퍼레이션을 제공한다.

2.2 소스 컨트롤러

소스 컨트롤러는 소스 코드에 정지점의 설정과 해지, 재실행, 한 라인씩 실행, 변수값 참조등 디버깅 과정에서 필요한 기능을 제공한다.

2.3 디버거 데몬

디버거 데몬(Debugger Daemon)은 목적 프로그램이 있는 원격 호스트에서 수행되는 컴포넌트로서 두 가지의 기능을 수행한다. 첫째로 서버 매니저로부터의 연결 요청을 받아들여 해당 태스크를 생성시킨다. 둘째로 생성시킨 태스크의 실행 모드가 디버그 모드일 경우 JaNeC 데몬 대신 태스크와 연결되어 마치 JaNeC 데몬인 것처럼 수행된다.

2.4 서버 매니저

서버 매니저(Server Manager)는 디버거측 컴포넌트로써 각 서버들을 호스트 리스트에 등록하고 각 호스트들의 태스크들을 태스크 리스트에 등록하여 병렬 환경을 설정하는 역할을 수행한다.

2.5 트레이스 파일

트레이스 파일(trace file)은 실행 순서에 영향을 줄 수 있는 요소, 즉 다른 태스크들로부터의 메시지가 기록된다. 트레이스 파일은 각 태스크마다 하나씩 존재하게 되며 재실행시 디버거 데몬에 의해서 읽혀져 태스크의 입력 메시지로 전달된다. 트레이스 파일의 이름은 호스트이름과 실행 파일 이름, 그리고 사용자 이름으로 구성되어 있다.

2.6 JaNeC에 적용

그림 2는 디버거가 JaNeC 애플리케이션 디버깅을 위해 네 개의 호스트에 연결되어 있는 그림이다. 서버 매니저에 의해서 각각의 호스트의 JaNeC 데몬과 디버거 데몬에 연결이 된다. 태스크를 디버깅하기 위해서는 서버 매니저에서 실행 모드를 디버그로 하여 디버거 데몬에 태스크 생성 메시지를 보내면 디버거 데몬은 debug 옵션을 주고 태스크를 생성하고 출력되어져 나오는 passwd를 서버 매니저에게 보낸다. 서버 매니저는 passwd를 입력 값으로 각각의 호스트의 태스크를 디버깅하기 위한 RemoteDebugger 쓰레드가 생성되어 디버깅 작업을 하게 된다.

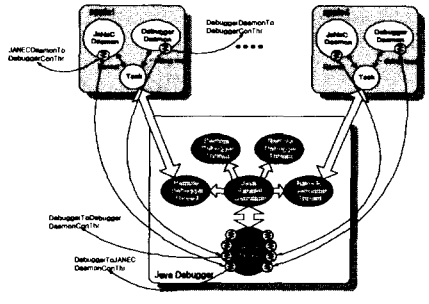

그림 2 디버깅시의 전체 구조도

3. 구 현

본 논문에서 구현한 재실행 자바 디버거는 하나의 태스크의 디버깅을 담당하는 RemoteDebugger를 쓰레드로 구현한다. 이렇게 구현된 RemoteDebugger는 쓰레드 윈도우와 소스 윈도우를 가지고 목적 프로그램의 정보를 얻고 디버깅 작업을 할 수 있다. 정지점에 도착 시와 정지 후 다시 시작시킬 때, RemoteDebugger 쓰레드는 하나의 쓰레드 그룹처럼 관리된다. 따라서 각각의 태스크에서 발생한 쓰레드들도 해당 태스크의 쓰레드 윈도우에서 제어가 가능하고 태스크 스케줄러에서는 각 태스크별로 제어가 가능하다. GUI는 재실행 자바 디버거를 메인 프레임으로 하고 하나의 목적 프로그램에 접속할 때마다 쓰레드 윈도우와 소스 윈도우 내부 프레임을 각각 하나씩 갖게 설계한다.

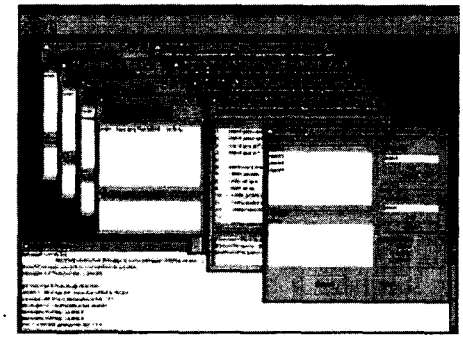

그림 3 자바병렬 디버거 전체화면

그림 3은 자바 병렬 디버거가 그림 2에서와 같은 환경에서 네 대 각각의 호스트의 태스크들에 연결된 상태를 보여준다. 각 태스크마다 쓰레드 윈도우와 소스 코드 윈도우를가지게된다

쓰레드 컨트롤 윈도우는 사용자 프로그램이 수행중에 발생하는 쓰레드를 나타낸다. 쓰레드 컨트롤 윈

도우에서는 쓰레드에 대한 suspend, resume 등의 제어 기능을 제공한다. 소스 코드 윈도우는 수행 중인 Java 프로그램 Generator의 소스코드를 나타낸다. 소스코드를 라인 별로 실행시키려면 우선 정지점을 설정해야 한다. 정지점을 설정하고 재시작(restart) 버튼을 누르면 소스 코드는 정지점을 만날 때까지 수행된다. 수행 중 정지점을 만나게 되면 main 쓰레드가 suspend 상태가 되며 수행이 중단된다. 이때 다음라인(next) 버튼을 누르면 한라인씩 실행되고, 계속(continue) 버튼을 누르면 정지점이 설정된 라인의 다음 라인부터 실행이 계속된다.

디버거의 서버 매니지는 현재 apple1부터 apple4까지 네 대의 호스트에 연결되어 있다. 이 윈도우에서 호스트와 태스크 그리고 실행모드를 선택하여 connect 버튼을 누르면 해당 태스크에 연결되어 디버깅 작업을 처리할 수 있다. 실행 모드가 Normal인 경우 태스크는 JaNeC 데몬과 연결되어 수행이 되게 되고, 트레이스인 경우 JaNeC 데몬과 연결되어 실행 도중 재실행에 필요한 정보들을 트레이스 파일로 모으게 된다. 그리고 디버그 모드인 경우 디버거 데몬과 연결되어 실행이 된다.

4. 결론 및 향후 과제

본 논문에서는 JaNeC 병렬 프로그램을 위한 재실행 자바 디버거를 제시하였다.

이 병렬 디버거는 RemoteDebugger API를 이용하여 구현되어졌기 때문에 원격 호스트의 자바 프로그램에 대한 디버깅이 가능하며 하나의 목적 프로그램에 RemoteDebugger 쓰레드가 연결되어 프로그램을 디버깅하는 구조로 되어 있어서 자바 병렬 프로그램을 디버깅할 수 있다. 그리고 재실행시 실행 순서의 보장을 위해 태스크의 실행 모드를 세 가지를 두고 있다. 첫 번째, 일반적인 경우 태스크는 JaNeC 데몬에 연결되어 디버거와는 상관없이 태스크간의 메시지 교환에 의해서 수행된다. 두 번째, 트레이스 모드인 경우는 태스크는 JaNeC 데몬에 연결되어 태스크간의 메시지 교환이 있을 때마다 메시지가 태스크로 전달되기 전에 트레이스 파일에 먼저 기록을 한 후 태스크에 전달되어 수행을 하게 된다. 마지막으로 디버그 모드로 실행될 때는 태스크가 JaNeC 데몬에 연결되는 것이 아니라 디버거 데몬에 연결되고 디버거 데몬은 트레이스 모드에서 기록된 트레이스 파일을 읽어들여 JaNeC에서의 메시지 형태인 JMessage로 바꾸어 태스크에 전달하게 된다. 따라서 재실행시 실행 순서를 보장 할 수 있다.

향후 과제로는 쓰레드 사이의 deadlock 탐지가 있다. 자바 병렬 프로그램의 특성상 여러 개의 자바 가상머신 프로세스가 발생되고 각 자바 가상머신은 여러 개의 쓰레드를 가질 수 있다. 따라서 이들 사이에서 deadlock이 발생할 가능성이 있다. 그러나 비록 자바 가상머신 구현체는 자신의 코드 내에서는 deadlock을 피하도록 설계되어져 있지만 사용자의 프로그램을 deadlock에서 보호할 수는 없다. 따라서 디버거에서 deadlock을 탐지해 프로그래머에게 알려 주어야 한다.

참고 문헌

[1] L. Beguelin, J. J. Dongrarra, A Geist, and R.J.M.V.S. Sunderan, "Heterogeneous Network computing," in *Sixth SIAM Conference on Parallel Processing*, SIAM, 1993.

[2] 홍철의 외 2인, "MPI 내실행에서의 인과관계 정지점", 한국정보처리학회 98년 추계 학술발표논문집, 제5권 2호, pp.261-264

[3] Daniel B. Price, "New Techniques of Replay Debugging", 논문지, available http://wilma.cs.brown.edu/ugrad/uresearch /papers/1998/dp.html

[4] Bill Venners, *Inside the Java Virtual Machine*, O'REILLY August 25, 1997.

[5] Jojn T.Stasko, "The Path-Transition Paradigm : A Practical Methodology for Adding Animation to Program Interfaces," College of Computing Georgia Institute of Technology Atlanta, Ga 30332-280.

[6] Scott Oaks , Henry Wong, *Java Threads*, O'REILLY ,January 1997.

[7] Fabrizio Baiardi, Nicoletta DE Francesco, Gigliola Vaglini, "Debelopment of a Debugging for a Concurrent Language", IEEE Transactions on Software Engineering, Vol.SE.12, No.4, 1989

[8] Barton P. Miller, Jong-Deok Choi, "A Mechaniam for Efficient Debugging of Parallel Programs", Technical Report TR754, University of Wisconsin, Madison, Department of Computer Science, 1988.

[9] L. Lamport, "Time, Clocks and the Orderings of Events in a Distributed System", Communications of the ACM, 21(7), pp.558-565, July 1978.

[10] Joan M.Francioni Cherri M.Pancake, "High Performance Debugging Standards Effort", available http://www.ptools.org/hpdf/article.html

[11] Anh Nguyen-Tuong and Andrew S. Grimshaw, "Exploiting Sequential Debuggers in a Parallel Environment: An Introduction to the Mentat Assistant Debugger", http://www.cs.virgi nia.edu/~men tat/

[12] Parallel Tools Consortium Projects, "Lightweight Corefile Browser", http://www.ptools.org/projects/lcb/summary.html