시간성 튜플 도입을 통한 LINDA 개념의 확장

박영환[0]
한성대학교, 이공대학, 컴퓨터공학과
yhpark@hansung.ac.kr

Extension of LINDA concept through the introduction timed-tuple

Young Hwan Park[0]

Dept. of Computer Engineering Hansung University

요 약

본 논문에서는 LINDA 개념에 시간성 튜플 개념을 도입함으로써 보다 빠르고 보다 간단한 IPC 메커니즘을 커널속에 구현할 수 있음을 보였다. 이 새로운 개념에서는 정보의 교환은 기존의 튜플을 그대로 이용하고 IPC는 시간성 튜플을 이용하여 구현하였는데, 튜플을 이렇게 2가지 종류로 나눔으로써 정보의 교환에 이용되는 튜플은 IPC에 사용되는 필드를 갖고있지 않아도 되고, 반대로 IPC에 사용되는 시간성 튜플은 정보의 교환에 이용되는 필드를 갖고있지 않아도 된다. 결과적으로 기존의 LINDA 개념을 이용한 커널의 크기보다는 작아진 커널을 구현할 수 있다. 또한 정보의 교환이나 IPC를 위하여 읽혀지거나 쓰여지는 정보의 크기도 작아지므로 커널 서비스 시간을 단축할 수 있다. 그리고 IPC를 위해서는 시스템 tic을 세는 증감 연산이 필요한데, 기존의 LINDA 개념에서는 모든 튜플을 읽고 처리해야 하지만 시간성 튜플 개념을 이용한 IPC에서는 단지 시간성 튜플에만 증감 연산이 필요하다. 결과적으로 보다 효과적인 커널 서비스를 구현할 수 있다.

1. 서 론

병렬 프로그램 개발상의 어려움은 둘 이상의 병렬 프로세스간에 필요한 정보의 교환을 언제, 어디를 통하여 해야하는가를 항상 고려해야 한다는 것이다. 이러한 문제점을 극복하고자 예일대(Yale University)의 David Gelernter와 그 팀에 의하여 병렬 프로그램의 한 모델인 LINDA 개념이 제안되었으며, 이 LINDA 개념을 통하여 프로세스에서 시간적인 그리고 공간적인 문제를 분리해 냄으로써 순차적 프로그램의 개발에 들어가는 노력과 같은 정도의 노력으로 병렬 프로그램을 개발할 수 있게 되었다[1].

또한 LINDA 개념을 커널에 도입하여 IPC(Inter process communication), 프로세스 동기화 등을 구현한 병렬 커널도 개발되었다[3][4]. 이렇게 LINDA 개념을 커널에 도입할 때, 이 개념의 문제점과 한계점 등을 고려하여야 하는데, 첫째로 read()와 in() 프리미티브의 역전에 의하여 깨어나지 못하는 프로세스가 존재하는 데드락 상태에 빠질 수 있다는 문제점이다. 이러한 문제점은 read()와 in()을 하나로 묶어서 근본적으로 역전이 일어나지 않게 하여 극복하고자 하는 연구가 있어왔다[5]. 둘째로 프로세스의 관리(동기화 문제)가 LINDA 개념만으로는 쉽지 않다는 점이다. 즉 한가지의 튜플 스페이스를 이용하여 데이터의 교환과 프로세스의 동기화를 모두 처리해야 하는데 이 경우 튜플의 구조가 두 목적 모두를 위한 내용을 포함하게 됨으로써 복잡해 질 수 있다.

본 논문에서는 위의 두 문제를 해결하기 위하여 시간성 튜플과 시간성 튜플 스페이스를 새로 제안하고 기존 개념 속의 튜플 구조와 비교함으로써 보다 단순하고 효율적인 구조의 커널을 구성할 수 있음을 보이고자 한다.

이 후 본 논문은 2장에서 LINDA의 기본 개념과 구조를 설명하고 3장에서 시간성 튜플과 튜플 스페이스의 구조에 대하여 언급할 것이며 4장에서는 기존 LINDA 개념과의 비교를 통하여 단순성과 효율성을 보이고자 한다. 그리고 5장의 결론을 끝으로 본 논문 마치려 한다.

2. LINDA

LINDA는 병렬 프로그램 개발상의 어려움을 극복하고자 예일대(Yale University)의 David Gelernter와 그 팀에 의하여 제안된 병렬 프로그램의 한 모델이다[2]. LINDA 개념을 통하여 프로세스에서 시간적인 그리고 공간적인 문제를 분리해 냄으로써 순차적 프로그램의 개발에 들어가는 노력과 같은 정도의 노력으로 병렬 프로그램을 개발할 수 있게 되었다[1].

이 모델은 튜플(tuple)과 튜플-스페이스(tuple-space) 그리고 4개의 기본적인 프리미티브들로 이루어져 있다. 튜플은 각 튜플 고유의 키(key) 필드와 데이터의 집합

필드로 구성되어 있다. 그리고 튜플-스페이스는 튜플들로 구성되어 있는 일종의 공용 연상 기억장치(Shared Associative Memory)로, 이 튜플-스페이스에서는 주소가 아니라 바로 각 튜플의 고유 키를 통하여 원하는 튜플을 찾을 수 있다. 원하는 튜플을 쓰거나 읽기 위하여 out(), eval() 과 in(), read()라는 4개의 프리미티브가 사용된다.

2.1 out()과 eval()

out(key, data)을 통하여 하나의 튜플을 튜플-스페이스에 삽입할 수 있다. 그리고 이렇게 삽입하는 과정에서 이 튜플을 읽기 위하여 in()이나 read()를 실행시킨 프로세스가 있나 검사하여 이 들을 깨운 후 프로세스 레디 큐(ready queue)에 추가시킨다. eval(key, func(e), true)은 기본적으로 out()과 같은 일을 하나, 우선 func(e)를 실행하여 그 결과가 사실이면 그 결과를 데이터로 하여 튜플을 생성하고 이 튜플을 튜플-스페이스에 삽입시킨다.

2.2 in()과 read()

in() 과 read()는 튜플-스페이스에서 원하는 튜플을 읽는 프리미티브들이다. 그리고 또 다른 두 프리미티브의 공통점은 원하는 튜플을 찾을 수 없는 경우, 그 프리미티브를 실행시킨 프로세스가 수면상태에 들어가고, 그 후 원하는 튜플을 삽입하는 out() 이나 eval()에 의해서 깨어나 원하는 튜플을 읽게된다. 두 프리미티브의 차이점은 read()는 튜플을 읽은 후 그 튜플을 지우지 않고 남겨놓는 반면 in()은 원하는 튜플을 읽은 후 그 튜플을 튜플-스페이스에서 지워버린다.

2.3 장점

LINDA 개념에서 프로세스로부터 시간적인 그리고 공간적인 문제를 분리해 냄으로써 순차적 프로그램의 개발에 들어가는 노력과 같은 정도의 노력으로 병렬 프로그램을 개발할 수 있다고 하였는데, 이러한 특성은 바로 튜플-스페이스와 위의 4가지 프리미티브 덕분이다. 즉 병렬처리에서 꼭 필요한 프로세스간 통신을 수행하는데 있어 필요한 정보의 입,출력이 주소가 아닌 키를 매개로 이루어지므로, 여러 개의 프로세스 개발 시 데이터의 공간적인 위치 문제를 고려함이 없이 프로그램을 할 수 있다. 또한 원하는 튜플이 없으면 프로세스가 자동적으로 수면상태에 들어가고, 그 원하던 튜플이 삽입되면 또한 자동적으로 깨어나게 됨으로써 프로세스간의 동기화를 고려하지 않고 병렬 프로그램을 개발할 수 있게 된다. 결과적으로 LINDA 개념을 통하여 병렬 응용프로그램을 개발함에 있어 공간적인 그리고 시간적인 제약을 뛰어넘을 수 있게 된다.

3. 시간성 튜플

기존의 튜플을 이용하여 데이터 교환을 위한 IPC를 커널 속에 구현하는 것은 직관적이고 쉬운 작업이다. 데이터를 주고자 하는 프로세스는 out()이라는 프리미티브를 통하여 정보를 필요로하는 프로세스에 직접 전달하거나 튜플 스페이스에 저장하고 하던 일을 계속 처리하면 되고 데이터를 받고자 하는 프로세스는 튜플 스페이스에서 정보를 찾아서 읽어 오거나 없는 경우 정보가 도착할 때까지 기다리면 된다. 이러한 과정 속에 프로세스 동기화 과정의 일부가 내재되어 있지만 사용자가 직접 하나의 프로세스를 대기시키거나 깨울 수 있는 시스템 콜을 기존의 LINDA 개념만으로 구현하기는 쉽지 않다. 이러한 문제를 데이터 교환을 위한 기존의 튜플과 프로세스 동기화를 위한 시간성 튜플로 나누어 해결할 수 있다.

3.1 시간성 튜플의 필요성

기존의 튜플 하나만을 이용하여 데이터의 교환과 프로세스 동기화를 동시에 구현하려 할 경우 튜플의 구조가 복잡해 질 수 있다. 즉 하나의 튜플에 데이터 교환을 위한 필드와 동기화를 위한 필드가 모두 포함되어 있어야 한다. 이렇게 되면 데이터 교환에 이용되는 튜플이 읽히거나 쓰일 때 전혀 사용되지 않는 필드가 같이 만들어지고, 이동되고, 삭제되거나 처리되어야 하므로 커널의 크기가 그 만큼 커지고 처리시간이 지연되게 된다. 또한 프로세스 동기화는 시간과 밀접한 관계가 있으므로 시스템의 시간 흐름을 모니터 하기 위하여 시스템 tic을 세는 카운터 필드가 존재해야 한다. 그리고 매 tic 마다 모든 튜플에 존재하는 이 카운터 필드의 tic 값을 조정해 주어야 한다. 따라서 데이터 교환에 사용되는 튜플도 매번 모두 읽혀져야 한다. 이러한 필요 없는 오퍼레이션은 커널의 처리 속도를 감소시키거나 오류를 일으키는 한 요인이 될 수 있다.

3.2 시간성 튜플의 구조

시간성 튜플도 일반적인 튜플과 같이 키에 의해서 접근되어지는데, 이 시간성 튜플의 구조는 아래와 같다.

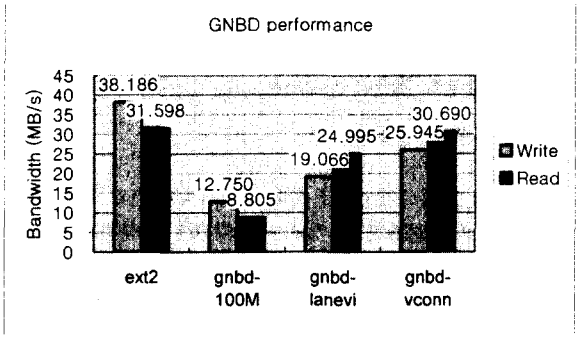
```
struct TTuple{
   int rtn_value;      /* synchronization result */
   int counter;        /* tic counter            */
}
```

rtn_value 필드는 프로세스간 동기화 결과를 저장하기 위한 필드로 counter 값이 이미 0 이면 아무런 변화가 없고 0 보다 크면 OK(=1) 값을 갖게된다. 수면 상태에 있는 프로세스가 깨어날 시간에 대한 정보는 counter 필드에 저장된다. 그리고 매 tic 마다 counter 필드의 값은 1씩 줄어들게 되며, 이 값이 0이 되

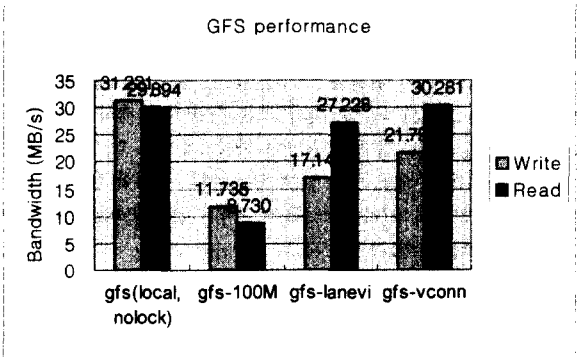
가 동작하지 않는 경우가 발생하였다. Bottom half를 사용하면 핸들러의 수행이 약간 지연될 수 있으나 메시지를 잃는 경우를 방지할 수 있다.

4. 성능평가

실험 장비는 Intel 815EP 메인보드를 사용하는 두 대의 LINUX(커널 버전2.2.18) 서버를 사용하였다. 각 서버는 256KB L2 캐쉬와 512MB 메인 메모리를 가진 Pentium III-1GHz CPU로 구성되어 있다. 두 서버는 두 장의 cLAN100 네트워크 어뎁터를, 별도의 스위치를 사용하지 않고, 바로 연결하였다. 디스크는 IDE UDMA66 인터페이스의 MAXTOR 40G이고 디스크의 성능을 측정하는 도구는 bonnie++ 1.0을 사용하였다.



[그림 4] GNBD의 성능



[그림 5] GFS의 성능

그림4는 GNBD/VCONN에 EXT2 파일시스템을 생성한 후 bonnie++를 사용하여 성능을 측정한 결과이다. VCONN로 통신했을 때 LANEVI보다 읽기가 약 22.7%, 쓰기가 약 30.7% 향상되었다. 읽기의 경우 디스크를 직접 읽을 때와 비슷한 성능을 보일 정도로 우수한 성능을 보인다. 그림5에서 보듯이, 로칼 GFS 파일시스템에서 성능을 측정한 결과도 VCONN을 사용하는 것이 LANEVI를 사용하는 것보다 좋은 성능을 보인다. LANEVI는 아직 불안정하게 동작하기 때문에 안정성 면에서도 VCONN이 우수하다.

GNBD의 통신이 요구(request)/응답(response) 형태이므로 지연시간(latency)이 우수한 VCONN을 사용하는 것이 성능을 향상시키는 것으로 판단된다. 메시지 크기가 22바이트일 때 지연시간이 VCONN은 약 17us, LANEVI는 약 68us이고, 4KB일 때는 각각 70us, 200us로서 VCONN이 월등히 우수함을 알 수 있다. VCONN의 지연시간은 사용자수준에서 측정한 것과 비슷한 결과로서 KVIPL을 사용하더라도 성능저하가 거의 없다는 것을 알 수 있다.

5. 결론

본 논문은 VIA상에서 GNBD를 위한 고속 통신 계층(VCONN)을 제안하였다. TCP/IP를 사용하는 LANEVI 대신에 VIA상에서 소켓 인터페이스를 지원하여 낮은 지연시간, 높은 대역폭의 통신을 가능하게 했을 뿐만 아니라 GNBD 코드를 거의 수정하지 않고 VIA 통신을 이용할 수 있도록 하였다. 또한 커널 수준에서 KVIPL을 확장하여 다른 커널 모듈이 VIA 통신을 사용할 수 있는 길을 제공하였고, VIA가 사용자 수준의 통신을 위한 구조를 가지고 있지만 필요에 따라 커널에서 사용하더라도 문제가 없음을 확인하였다. GNBD의 통신모듈을 TCP/IP/LANEVI에서 VCONN으로 교체하였으나 GNBD 인터페이스는 전혀 수정하지 않았기 때문에 기존의 GNBD 관련 도구들을 그대로 사용할 수 있다.

앞으로 VCONN을 확장하고 완전한 기능을 가진 커널 수준 소켓 인터페이스(KSOVIA)를 제공할 계획이 있고, 그 소켓 인터페이스를 사용자수준까지 지원하여 사용자 수준에서 소켓으로 작성한 응용 프로그램이 KSOVIA를 통하여 VIA를 이용할 수 있을 것이다.

6. 참고문헌

[1] Andrew Barry, at el., "Implementing Journaling in a Linux Shared Disk File System," Sistina Software, Inc.

[2] Jin-Soo Kim, Kangho Kim, and Sung-In Jung, "Building a High Performance Communication Layer Over Virtual Interface Architecture on Linux Clusters," Proceedings of the 15th ACM International Conference on Supercomputing (ACM ICS '01), pp.335-347, Sorrento, Italy, June 2001.

[3] "VI Architecture Software Developer's Guide," Giganet, 1999.

[4] Kevin Duncan, "Fiber channel and Gigabit Ethernet: A Look at Technology for Storage Networking Solutions," University of Minnesota Fiber Channel Group, Minneapolis, MN May 14, 2001.

[5] R. Buyya, editor. High Performance Cluster Computing: Architecture and Systems. Prentice Hall, Inc., 1999.

[6] B. Chun, A. Mainwaring, and D. Culler. Virtual Networking Transport Protocols for Myrinet. IEEE Micro, 31(1):53-63, Jan. 1998.

[7] Philip Buonadonna, Andrew Geweke, and David E. Culler, "An Implementation and Analysis of the Virtual Interface Architecture," In Proc. SC '98, 1998.

[8] M-VIA, "http://www.nersc.gov/research/FTG/via/

[9] G. Pfister. In Search of Clusters, 2nd ed., Prentice Hall, Inc., 1998.

[10] Alessandro Rubini, Linux Device Drivers, 1st ed., O'Reilly & Associates, Inc., 1998.