리눅스 클러스터를 위한 커널 수준 통신 시스템의 설계 및 구현

박동식+ ○ 박성용+
+서강대학교 컴퓨터학과
이장선+ 오상규+
+(주)매크로임펙트
bilbo@dcclab.sogang.ac.kr, parksy@ccs.sogang.ac.kr,
sunny@macroimpact.com, sgoh@macroimpact.com

Design & Implementation of Kernel Level Communication System for Linux Cluster

Park, Dong-sik+ ○ Park, Sungyong+
+Dept. of Computer Science, Sogang University
Lee, Jang-Sun+ Oh, Sang-Gue+
+MacroImpact, Inc

요 약

본 논문에서는 리눅스 클러스터에서 커널 수준의 응용 프로그램 개발을 위한 통신 모듈인 KCCM(Kernel level Cluster Communication Module)에 대한 구조를 기술한다. KCCM은 비동기 통신을 지원하기 위한 응용 프로그램 인터페이스와 송수신(Send/ Receive)형태의 동기 통신을 지원하기 위한 응용 프로그램 인터페이스를 함께 제공하며, 다른 플랫폼으로의 포팅을 고려하여 소켓 인터페이스를 이용해 구현되었다. 또한 장애 상황에서도 서비스를 계속하고 장애를 복구할 수 있도록 설계되어 커널 수준의 고 가용성 클러스터 응용 프로그램을 개발하는데 유용하다.

1. 서론

아바론 클러스터의 성공으로 인하여 클러스터에 대한 관심은 폭발적으로 증가하게 되었다. 클러스터를 이용하여 슈퍼컴퓨터를 만들수 있게 된 것은 저렴하고 고성능의 범용 통신 장치들을 이용할 수 있게 되었기 때문이다.

클러스터가 범용의 슈퍼컴퓨터의 역할을 대신하기 위해서는 싱글 시스템 이미지를 제공하는 서비스와 응용 프로그램들이 필요하며 이런 싱글 시스템 이미지를 만들기 위해서는 서비스나 응용 프로그램 개발을 위해서 사용할 수 있는 통신 시스템이 필요하다. 이런 통신 시스템은 일반적으로 다음과 같은 요구사항을 가지고 있다.

첫째로 클러스터를 위한 통신 시스템은 높은 지연시간의 오버헤드를 줄일 수 있어야 최근의 통신 시스템들에서는 비동기 통신 패러다임을 많이 사용하고 있다.[1]

둘째로 응용 프로그램을 여러 가지 다른 플랫폼에 포팅하기 쉬워야 한다. 클러스터 응용프로그램은 보통 작은 규모의 클러스터에서 제작한다. 이런 개발 환경이 가능하게 하기 위해서는 클러스터용 통신 시스템이 하드웨어의 종류나 규모가 다를 때에도 쉽게 이식할 수 있는 특성을 가지는 것이 필요하다.

셋째로 클러스터를 위한 통신 시스템은 장애 복구 기능을 가져야 한다. 한 개 노드의 장애 때문에 클러스터 시스템의 동작이 멈추게 되면 손해가 커질 수 있기 때문에 클러스터용 통신 시스템은 장애가 생기더라도 전체 시스템이 멈추지 않고 계속 동작하여 그 노드가 정상이 되면 다시 복구되어 이전과 같은 상태로 동작할 수 있어야 한다.

넷째로 사용이 쉬워야 한다. 고성능을 제공하는 저수준의 통신 시스템들은 사용이 어렵기 때문에 응용 프로그램을 제작하기가 힘든 단점이 있다. 클러스터를 위한 통신 시스템은 기존 응용 프로그램들이나 알고리즘들을 구현하기 쉬워야 한다.

기존의 클러스터를 위한 유저수준의 통신 시스템으로 PVM(Parallel Virtual Machine)[2]이나 MPI(Message Passing Interface)[3]가 있다. 이런 통신 시스템들은 유저 수준에서 구현되었기 때문에 커널 수준에서만 제공할 수 있는 많은 기능들을 클러스터 시스템에서는 사용할 수 없다.

본 연구에서는 커널에서 이용할 수 있는 편리하고 쉬운 인터페이스를 가진 통신 시스템인 KCCM(Kernel-level Cluster Communication System)을 설계하고 구현하고자 하였다. KCCM은 통신을 사용하는 커널수준의 서비스 개발을 편하게 할 수 있고, 기존의 유저수준에서 검증된 알고리즘을 커널로 옮길 때 어려움 없이 구현할 수 있으며 이 통신 시스템 하부 인터페이스를 투명하게 만들어 더 좋은 저 수준의 네트워크 시스템을 도입할 경우 그 효율을 어플리케이션에게까지 전달해 줄 수 있는 구조와 비동기 통신을 지원하도록 설계하였다. 또 장애 상황에서 대처를 할 수 있어야 하며 사용이 쉽도록 설계하였다.

본 논문의 구성은 다음과 같다. 2절에서는 우리가 개발한 통신 시스템의 구조를 기능에 따라 설명한다. 3절에서는 만들어진 시스템의 성능을 측정하여 비교해보며 4절에서는 본 연구의 결론

을 내리고 추후 연구 과제에 관하여 논의한다.

2. 통신 시스템의 구조

리눅스 클러스터를 위한 커널 수준 통신 시스템인 KCCM(Kernel-level Cluster Communication Module)은 TCP를 기반으로하는 시스템으로 다중 쓰레드(Multithread) 방식으로 설계되었다. 그림 1은 KCCM의 전체적인 구조를 설명하고 있다.

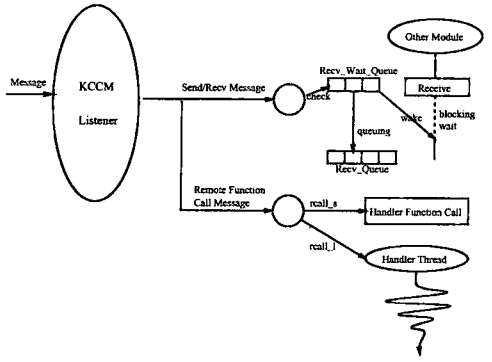

그림 1: KCCM의 구조

KCCM은 리스너 쓰레드를 중심으로 동작하게 되어 있다. 리스너 쓰레드는 항상 메시지를 기다리는 상태를 유지하고 있으며 메시지 패킷이 도착하면 패킷을 분석하여 패킷의 종류에 따라 처리하고 다음 처리를 위해 다시 기다리는 구조로 되어 있다.

메시지의 종류는 크게 송수신(Send/Receive) 메시지와 원격 함수 호출 메시지로 구분될 수 있으며 송수신 메시지는 모듈의 상황에 따라 큐잉하거나, 해당하는 프로세스에 전달해주고 원격 함수 호출 메시지는 메시지를 처리하는 시간에 따라 딜레이를 최소로 하기 위해 short와 long으로 나누어 처리한다. 어떤 프로세스 또는 커널 흐름(flow)에서 데이터를 받기 위해 수신을 요청하면 이미 도착했는지를 확인하기 위해 수신 큐를 살펴본 후 도착하지 않았다면 자는(Sleep) 상태에서 기다리게 되며 도착한 경우에는 수신 큐에서 가져간다. 자는 상태에서 기다리는 프로세스가 있다는 것을 리스너 쓰레드에 알리기 위해 별도의 큐(수신 대기 큐)를 구성하여 메시지가 도착하면 곧바로 깨워서 전달할 수 있게 한다.

2개의 커널 흐름(flow)이 송신과 수신을 통해 메시지를 받은 경우 두 가지 상황이 발생할 수 있다. 첫째는 그림2에서처럼 보낸 쪽에서 미리 보내놓고 상대편 노드의 큐에 저장된 상태에서 상대편 노드에서 수신을 부르면 커널의 흐름은 멈추지 않고 바로 다음으로 넘어가게 된다. 큐에 메시지가 2개 이상일 경우에는 먼저 도착한 메시지를 전달해준다. 두 번째는 그림3에서처럼 다른 상황으로 한 개의 흐름이 수신을 부른 상태로 큐를 살펴서 도착한 메시지가 없긴 때문에 자는 상태로 들어가는 상황이 있을 수 있다. 이 경우 수신을 부른 쪽에서는 웨이팅 큐에 자기가 자는 상태에 있으며 어떤 메시지를 기다리는지를 표시해둔다. 리스너는 상

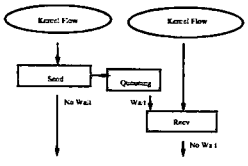

그림 2: Queuing 하는 경우

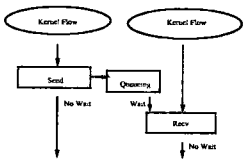

그림 3: 직접 전달되는 경우

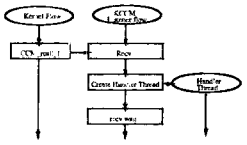

그림 4: CCM_rcall_l

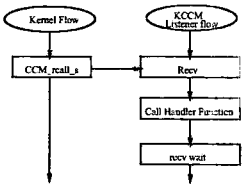

그림 5: CCM_rcall_s

대편 노드에서 메시지가 도착하면 웨이팅 큐를 보고 기다리던 흐름을 깨워서 그 메시지를 전달한다.

원격 함수 호출을 할 경우 파라미터와 함께 상대편 노드로 메시지가 전달되게 되어 있다. 상대편 노드에 메시지가 도착하면 리스너는 메시지 헤더를 보고 타입을 확인한다. Short 타입의 원격 함수 호출 메시지가 도착하면 적합한 핸들러 함수를 찾은 후 함수 호출로 수행한다. 수행이 끝나면 다시 리스너 루틴으로 돌아가 메시지를 기다리는 상태가 된다. 그렇기 때문에 Short 타입 메시지의 경우 핸들러 안에서 오래 지체되면 리스너가 다음 메시지를 처리하지 못하기 때문에 문제가 될 수 있다.

원격 함수 호출은 Long 타입과 Short 타입 두 가지가 있다. Long 타입은 핸들러가 쓰레드 형태로 수행되며 그림4에서처럼 리스너는 원격 함수 호출 메시지를 받으면 곧바로 핸들러 쓰레드를 생성시키고 다음 메시지를 기다리며 생성된 핸들러는 등록된 핸들러를 수행하도록 구현하였다. Short 타입은 그림5에서처럼 리스너 안에서 핸들러 함수를 수행하도록 구현 하였으며 핸들러 수행이 끝나면 다시 다음 메시지를 처리한다.

2.1 접속 과정의 구현

KCCM은 TCP위에서 만들어졌기 때문에 접속을 관리하는 것이 필요하다. TCP는 접속을 하는 곳과 접속을 받는 곳이 나뉘어 있기 때문에 두 노드가 접속을 형성할 때 항상 한쪽이 다른 한 쪽을 기다리도록 만들어져야 한다. KCCM이 접속을 관리하는 과정은 크게 처음 부팅하여 KCCM을 위한 모든 노드로의 접속들을 형성하는 것과 동작 중인 상태에서 한 노드 이상이 문제가 생겨 접속을 다시 형성하는 복구(Recovery) 상황으로 구성된다.

모든 노드로의 접속은 항상 노드 번호가 낮은 노드에서 접속을 시도하고 노드 번호가 높은 노드에서 접속을 기다리는 원칙으로 모든 노드에서 동시에 노드 순서대로 접속 형성을 시도하는 방법을 통해 전체 접속 상태(Fully Connection State)를 형성한다. 그 알고리즘은 다음과 같으며 실제 노드의 갯수 - 1 번의 접속 과정이 필요하다.

```
for (i=0; i<MAX_NODE; i++) {
        if ( i>Node_id ) Connect(i);
        if ( i<Node_id ) Listen(i);
        if ( i==Node_id ) Continue(i);
}
```

네트워크 이상이나 컴퓨터 이상 또는 관리상의 이유로 한 노드 이상이 통신을 할 수 없게 된 경우 문제가 되는 노드를 복구하는 과정에서 통신 시스템의 접속도 다시 형성해야 할 필요가 있다. 접속 관리자는 그림6처럼 노드의 번호에 따라 끊어진 접속에 대해 새로운 접속을 시도하여 전체 접속 상태를 회복한다.

접속을 복구하는 과정은 크게 3단계로 이루어진다. 첫 단계는 끊어진 접속이나 문제가 생긴 노드를 확인하는 과정으로 메니저는 자체의 알고리즘에 의해 끊어진 접속을 확인한다. 두 번째 단계는 끊어진 접속의 소켓을 제거하는 과정으로 문제가 생긴 노드에 대한 통신을 더이상 진행하지 못하도록 막는 역할을 한다. 세 번째 단계는 다시 접속을 복구하는 과정으로 각 노드는 끊어진 접속에 대해 노드 번호에 따라 새로운 접속을 시도한다. 이 과

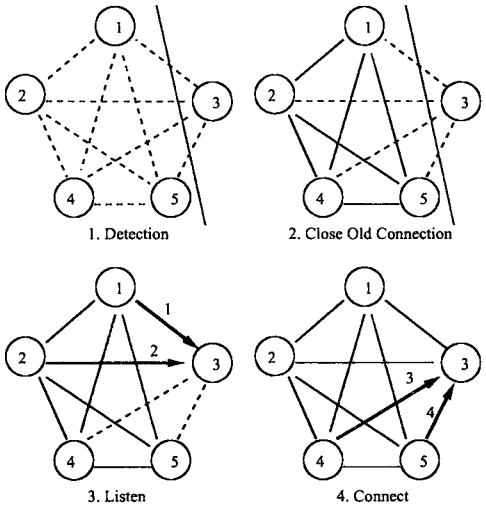

그림 6: 접속 복구 과정

정은 크게 접속을 기다리는 것과 접속을 시도하는 두가지 단계로 이루어진다.

3. 성능평가

본 연구에서는 하드웨어 환경으로 256KB캐쉬를 가진 펜티움 800MHz 대칭형 멀티 프로세서 시스템과 네트워크 인터페이스로 3c905B 100Mbps 이더넷을 이용하였으며, 소프트웨어 환경으로 리눅스 커널 2.4.14에서 egcs2.91.66으로 최적화 옵션 없이 실험하였다.

3.1 원격 함수 호출

KCCM의 원격 함수 호출은 타겟 노드에서 등록된 함수를 수행시킨다. RPC가 등록된 함수의 리턴 값을 다시 돌려주는 동기식 동작을 하는데 비해서 KCCM은 일방적으로 타겟 노드에서 원격 함수를 수행시키는 비동기식 동작을 한다. 그래서 본 논문에서는 호출될 원격 함수 내부에서 리턴 값을 보내주는 것으로 비동기 동작을 대신하여 RPC와 성능을 비교하였다.

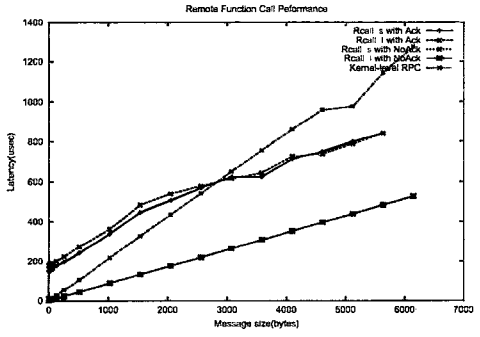

그림 7: 원격 함수 호출의 성능 측정 결과

그림7에서 보면 알 수 있듯이 비동기 원격 함수 호출에 비해 동기 원격 함수 호출이 월등히 빠른 것을 볼 수 있으며 이것은 매번 호출한 노드에 메시지를 보내지 않기 때문에 일어나는 당연한 결과이다. 리턴 값을 받지 않아도 되거나 일정 주기로 확인만 하면 되는 어플리케이션을 작성할 때 비동기 원격 함수 호출을 이용하면 성능향상에 도움이 될 수 있다. RPC와 KCCM을 비교할 때 3000바이트 이상의 메시지를 보낼 때 KCCM이 더 좋은 성능을 보이는 것을 알 수 있다. 또한 RPC의 경우는 작은 메시지에 대하여 최적화가 잘 되어 있어서 좋은 성능을 보이는 것을 알 수 있다.

3.2 Short 타입 원격 함수 호출

KCCM에는 원격 함수의 수행시간이 짧아서 쓰레드로 수행시킬 경우 쓰레드 생성 오버헤드가 원격 함수 수행 오버헤드보다 클 경우나 쓰레드 생성으로부터 수행까지의 시간 만큼의 지연시간을 줄이기 위해 Short 타입의 원격 함수 호출을 지원한다. 원격 함수의 수행 시간이 짧을 경우 Long 타입과 Short 타입의 원격 함수 호출에 걸리는 시간을 측정하여 Short 타입 원격 함수 호출의 필요성을 검증해 보았다.

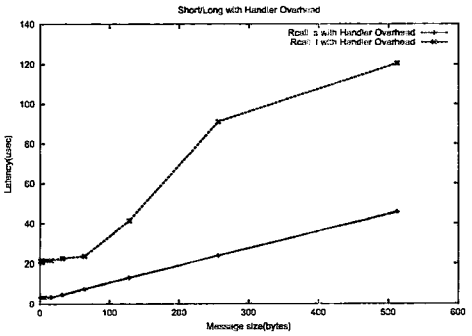

그림 8: Short,Long 타입 원격 함수 호출의 성능 측정 결과

그림 8에서 보면 알 수 있듯이 원격 함수의 수행 시간이 짧을 경우 쓰레드 오버헤드 보다 함수 호출의 오버헤드가 작기 때문에 Short타입으로 수행할 때 시간이 줄어드는 것을 확인할 수 있다.

4. 결론

본 논문에서는 클러스터 환경에서 커널수준의 어플리케이션이나 서비스 개발을 위한 쉽고 효율적이며 유연한 통신 시스템을 설계하고 구현하였으며 그 성능에 대해 논의 하였다. 본 논문에서는 커널 수준의 통신 시스템을 비동기 통신과 동기 통신을 함께 사용할 수 있도록 설계하였으며, 사용자의 응용 프로그램 제작을 용이하게 하기 위해 단순한 원격 함수 호출의 인터페이스를 제공하였다. 또한 통신 중에 생길 수 있는 장애에 대처하기 위해 접속을 관리할 수 있는 방법을 제공하였다. 본 논문에서 제안한 통신 시스템을 사용할 경우 커널 수준의 응용 프로그램을 설계하는데 유용할 것으로 기대된다.

참고 문헌

[1] A. Mainwaring, D.E. Culler, S.C. Goldstein and K.E. Schauser. *"Active Messages: a Mechanism for Integrated Communication and Computation"*. Proc. of the 19th ISCA, pp 3-4 ,1992.

[2] *"Parallel Virtual Machine"*. http://www.epm.ornl.gov/pvm/ 2001

[3] E. Lusk, B. Saphir, M. Snir. *"MPI-2: Extensions to the Message-Passing Interface"*. MPI Standard 2.0, 1997.