VIA를 이용한 RPC 성능 개선

김강호° 김진수 정성인
한국전자통신연구원 리눅스연구팀
{khk, jinsookim, sijung}@etri.re.kr

Improving Performance of the RPC Using the VIA

Kangho Kim° Jin-Soo Kim Sungin Jung
Linux Research Team, ETRI

요 약

최근에 클러스터 시스템 사용이 보편화되어 가고 있지만 클러스터를 구성하고 있는 노드 사이의 통신이 여전히 전체 성능 향상의 병목요인으로 지적되고 있다. 현재 클러스터 시스템의 노드간 통신은 TCP 프로토콜을 이용하고 있는데, 동질적이고 전송에러를 무시할 수 있는 클러스터 통신망에는 적합하지 않다. TCP의 단점을 극복하기 위하여 클러스터를 위한 다양한 사용자 수준 인터페이스가 제안되고 구현되었다. 이 중 Intel, Compaq, Microsoft가 주축이 되어 정의한 VIA는 SAN 환경에 적합하도록 기존의 소프트웨어 오버헤드를 줄인 사용자 수준의 통신 프로토콜이다. 본 논문에서는 현재 리눅스에서 사용가능한 사용자 영역 RPC 구조를 살펴보고, SOVIA(Socket/VIA)를 하부 전송 프로토콜로 사용하여 RPC의 성능을 개선하는 방법을 제안한다. 개선한 RPC는 VIA의 성능을 사용하면서 RPC 프로그래밍에는 변화가 없으므로 VIA를 지원하는 분산 프로그래밍 환경으로 적합하다.

1. 서론

근래 들어 리눅스를 이용한 클러스터 시스템의 구성이 보편화되어 가고 있지만, 클러스터를 구성하고 있는 노드 사이의 통신이 여전히 전체 성능 향상의 병목요인으로 지적되고 있다. 현재 대부분의 리눅스 클러스터 시스템은 노드간 통신을 위하여, WAN(Wide-Area Network) 환경에서 양극단간의 신뢰성있는 패킷 송수신을 위해 고안된 TCP/IP 프로토콜을 이용하고 있다. 그러나 클러스터 내부의 통신망은 동질적이고, 전송 에러를 무시할 수 있을 만큼 신뢰성이 높기 때문에, 기존의 TCP/IP 프로토콜 내부에서 수행되는 여러 불필요한 기능을 제거한 경량의 프로토콜을 사용하는 것이 바람직하다. 또한, 기존의 TCP/IP 프로토콜은 통신 프로토콜이 커널 내부에 존재하기 때문에 매 패킷 전송시마다 사용자 모드에서 커널 모드로의 문맥 교환이 수반되며, 사용자 주소 영역에 있는 데이터를 커널 내부로 복사한 후 전송해야 하는 단점이 있다. 이러한 TCP/IP 프로토콜의 단점을 극복하기 위하여 AM, BIP, FM, PM, U-Net, VMMC, VIA 등 클러스터를 위한 다양한 종류의 사용자 수준 인터페이스가 제안되고 구현되었다[1].

이 중 Intel, Compaq, Microsoft가 주축이 되어 정의한 VIA(Virtual Interface Architecture)[2]는 SAN(System Area Network) 환경에 적합하도록 기존의 소프트웨어 오버헤드를 줄인 사용자 수준의 통신 프로토콜로써, 새로운 산업계의 표준으로 대두되고 있다.

본 논문에서는 현재 리눅스 상에서 사용하는 RPC를 살펴보고, VIA[6,7,8]를 이용하여 RPC의 성능을 크게 향상시키는 방법을 제안한다.

2. RPC (Remote Procedure Call)

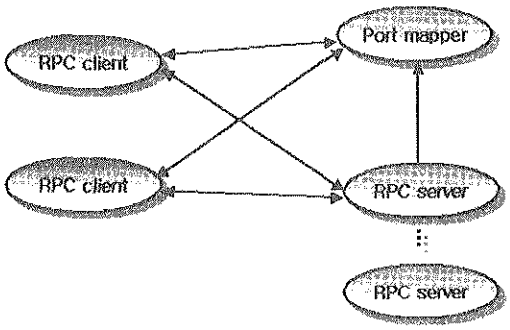
RPC는 전통적인 프로그래밍 언어에서 사용하는 함수 호출 기법을 많이 차용했다. RPC 모델은, 전통적인 프로그램과 동일한 절차적 추상화(procedural abstraction)를 사용하지만, 두 컴퓨터에 걸쳐있는 함수호출을 허용하여 분산 컴퓨팅을 용이하게 한다.

RPC 서비스는 그림1에서 보는 것과 같이 크게 3가지 구성요소로 이루어져 있다; 서버, 클라이언트, 포트 맵퍼(port mapper). 포트맵퍼는 미리 정해진 포트를 사용하는 일종의 RPC 프로그램으로 서버가 서비스번호와 사용하는 포트를 등록하는 곳이고, RPC서비스에 해당하는 포트 번호를 알려주는 곳이다. 클라이언트는 프로그램번호, 버전번호, 전송프로토콜 유형으로 포트 맵퍼에게 문의하여 지정한 서비스가 사용하는 포트를 알아내고, 그 포트로 프로시저의 번호와 인자를 전송하여 원격지 프로시저를 호출하는 순서로 서비스가 이루어진다. 포트번호를 얻은 후, 원격 호출은, (1) 인자를 메시지에 담고, (2) 메시지를 송신하고, (3) 서버에서 메시지를 받아 (4) 처리하고, (5) 결과를 응답 메시지에 담아 (6) 요청자에게 전달하고, (7) 요청자가 메시지를 수신해서 (8) 메시지를 해석하는 단계로 이루어진다.

RPC의 내부 구조는 기본 프로토콜인 TCP, UDP 위에서 XDR을 사용하는 형태이다. 통신 프로토콜은 필요에 따라 선택할 수 있다. XDR은 이기종간의 데이터 변환을 위한 소프트웨어 층(layer)이다.

프로그래머는 인터페이스 정의 파일을 작성하고 *rpcgen*이라는 도구를 사용하여 클라이언트 스텁(stub), 서버 스텁(stub), 헤더 파일을 생성한다. 클라이언트는 클라

이언트 스텁에, 서비는 서버 스텁에 맞추어서 하나의 프로그램을 작성하면, 클라이언트와 서버간의 통신은 스텁과 RPC 라이브러리에 의해서 처리된다.



[그림 1] RPC 서비스 구조

RPC 구현은 여러 종류가 있으나 우리는 리눅스에서 사용하는 *glibc-2.1.3*에 포함된 *sunrpc*를 대상으로 실험하였다.

3. VIA 통신

VIA를 사용하면 SAN 상에서 짧은 지연시간, 높은 대역폭으로 통신할 수 있지만, 주로 사용자 수준의 단일 메시지의 전송을 위한 최소한의 인터페이스만 제공한다. 그 인터페이스는 송수신 동기화, 흐름제어 등과 같은 기능이 없기 때문에 사용하기 어렵다. 그래서 응용프로그래머가 직접 VIA 인터페이스를 사용하는 것은 적합하지 않다. VIA API 위에 사용하기 쉽고, 친근한 통신계층이 필요하다. [7]에서는 VIA의 성능을 발휘하면서 쉽게 사용할 수 있는 메시지 전달 프리미티브를 제안하였다.

SOVIA는 VIA를 이용한 효율적인 메시지 전달 프리미티브[7]를 사용하여 소켓 인터페이스를 구현한 것이다[8]. 소켓은 다양한 프로토콜을 지원하지만 SOVIA는 TCP/IP 의미(semantic)를 가진 블락킹 모드의 인터페이스만을 구현하였다. VIA 인터페이스 구현이 사용자 수준에 있기 때문에 커널 수준에 구현된 소켓과 동일하게 행동하도록 구현하는 것은 매우 어렵다. 예를 들어, 동시 서버(concurrent server)와 같이 fork()를 사용할 경우는 고려하지 않고 있다.

지연시간(Latency) 그림 2는 현재 클러스터에서 실험한 전송프로토콜의 지연시간을 측정한 결과이다. 실험 플랫폼은 Intel L440GX+ 보드, Pentium III-500MHz CPU 및 리눅스 커널 2.2.16(UP)이 탑재되어 있는 PC이다. 지연시간은 메시지가 왕복하는데 걸리는 시간의 반으로 측정하였다.

예상하는 것처럼 FastEthernet을 사용한 TCP가 가장 긴 지연시간(75.5us)을 나타내고, VIA 인터페이스를 직접 사용한 NATIVE_VIA가 가장 짧은 지연시간(8.5us)을 나타낸다. Giganet이 지원하는 cLAN상의 소켓 인터페이스는 지연시간이 55us로 NATIVE_VIA의 성능에는 크게 못미치고 있다. 그 이유는 cLAN 소켓 인터페이스 구현이 cLAN에 최적화되지 않았고, 소켓의 다양한 요구사항을 반영하면서 성능을 희생한 것으로 생각된다. 이에 비해 사용자 수준에서, 제한된 기능의 소켓 인터페이스를 구현한 SOVIA는 NATIVE_VIA에 근접하는 성능(12us)을 보이고 있다. 위에 제시한 값은 4바이트를 전송할 때의 지연시간이다.

SOVIA의 전송속도를 측정한 결과도 동일한 경향을 보인다. NATIVE_VIA의 최대 전송속도는 메시지 크기가 32K바이트일 때 815Mbps이고, SOVIA의 전송속도는 NATIVE_VIA의 속도에 조금 못 미치는 것으로 측정된다.

제목

작성한 사람

이리 보기:
이 EPS 그림은 미리 보기 그림을 포함하지 않고
저장되었습니다.
설명:
이 EPS 그림은 포스트스크립트 프린터를
제외한 다른 프린터에서는
인쇄되지 않습니다

[그림 2] 지연시간

위에서 보여준 전송 프로토콜 중, RPC에 가장 적합한 것은 SOVIA라고 판단된다. NATIVE_VIA는 성능면에서는 우월하지만 RPC에 적용하기 어렵다. VIA API는 통신을 위한 최소 기능만 제공하고 있고, API가 소켓 API와 크게 다르기 때문에 RPC 라이브러리에 적용하려면 라이브러리 전체를 재작성해야 한다. cLAN을 사용한 TCP는 RPC 라이브러리를 전혀 수정하지 않고 사용할 수 있으나 역시 RPC에 적용하기 곤란하다. 전송지연시간과 대역폭이 기대수준 이하이므로 RPC에 적용하더라도 cLAN을 사용했을 때 기대하는 성능향상을 얻기 어렵다. SOVIA의 경우 NATIVE_VIA에 비해 약간의 성능 저하가 있을 뿐, 최소한의 소켓 인터페이스는 제공하므로 RPC 라이브러리를 수정을 최소화하면서 VIA를 사용할 수 있다.

SOVIA는 RPC/VIA를 구현하기 위해서 send(), recv(), connect() 등과 같은 기본 소켓 함수에 추가하여 select()와 poll()을 지원한다. RPC 서버측에서 소켓 입출력에 다중 송수신(multiplexing)하기 위해서 select()가 필요하다.

4. RPC/SOVIA

RPC 프로그래밍 모델에서는 프로그래머는 전송 프로토콜을 지정하고, 원격 프로시져의 인터페이스 정의로부터 생성된 코드를 사용하여 통신한다. 새로운 통신 프로토콜을 지원하기 위해서는 RPC 코드 구조에 따라 그 통신 프로토콜을 사용하는 라이브러리를 구축하고, 그 라이브러리를 사용하도록 하는 코드생성기를 제작해야 한다.

그래서 우리는 기존의 RPC 코드 중에서 TCP 모듈을 수정하여 SOVIA를 반영하였고, 수정된 모듈을 사용하도록 하는 코드생성기도 제작하였다. SOVIA를 의도적으로 TCP 의미에 맞추어 설계했기 때문에 TCP 모듈을 수정하

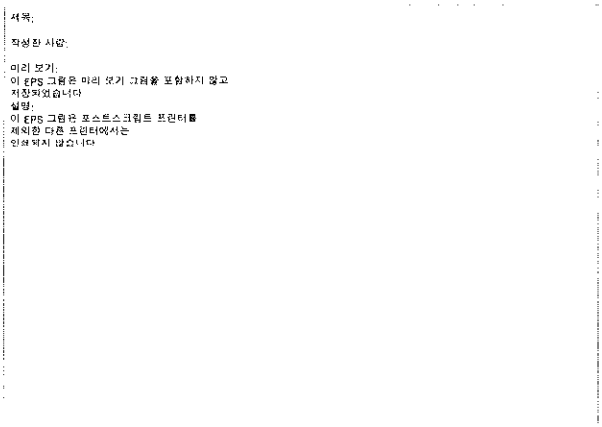
는 방법은 굉장히 간단하다. 기존 소켓 인터페이스 호출 함수를 SOVIA 함수로 대치하면 된다. 이렇게 하여, RPC 코드 수정을 최소화하면서 RPC 라이브러리에 VIA 프로토콜을 추가할 수 있었다. SOVIA를 구현하는 측면에서도 RPC에 맞추어진 소켓 인터페이스를 지원하는 것이 일반적인 소켓 인터페이스를 지원하는 것보다 훨씬 쉽다.

포트맵퍼와의 통신은 기존의 TCP 소켓을 사용하도록 한다. 포트맵퍼는 클라이언트 또는 서버가 실행될 때 한 번만 참조하는 서버이므로 통신 속도 향상이 필요하지 않은 부분이다.

RPC 클라이언트는 기존의 통신 프로토콜(TCP, UDP, unix domain)과 SOVIA를 병용할 수 있다. RPC 서버의 경우는 select 함수에서 SOVIA 소켓 디스크립터와 기존 소켓 디스크립터를 동시에 처리하지 못하기 때문에 RPC/VIA 서버는 별도로 제작되어야 한다

RPC/SOVIA와 RPC/TCP의 응답시간 VIA 지연시간을 측정한 플랫폼과 같은 곳에서 원격 함수 호출의 응답시간을 측정하였다. SOVIA를 RPC에 적용한 결과는 그림3에서 볼 수 있다. 경과시간(elapsed time)은 원격함수가 호출된 후 결과가 되돌아 올 때까지의 시간을 전달하는 인자의 크기별로 측정한 것이다. 성능 측정은 RPC/TCP(FastEthernet 기반), RPC/cLAN(cLAN을 사용한 TCP 기반), 그리고 RPC/SOVIA(SOVIA를 사용한 TCP 기반)를 대상으로 하였다.

전달되는 인자가 없이 원격함수를 호출하는 경우, 즉 인자크기가 0인 경우는 RPC/SOVIA의 응답시간이 36.3us, RPC/clan이 162us, RPC/TCP가 202us이다. RPC/SOVIA의 경우를 좀 더 자세히 관찰해 보면, 총 36.3us의 시간 중에서 클라이언트 측에서 메시지를 처리하는데 걸리는 시간이 약 1.7us, 통신하는데 걸리는 시간이 약 27.4us, 서버 실행시간이 약 7.2us이다. 인자의 크기가 0이지만 실제로 전달되는 메시지의 크기는, 보낼 때 44바이트, 받을 때 28바이트이다. 통신 시간은 이 메시지를 송수신하는데 걸리는 것으로 SOVIA에서는 각 12.9us, 12.6us가 걸린다. SOVIA에서 통신에 걸리는 시간이 25.5us인데 비해서 RPC/SOVIA의 통신 시간이 27.4us로 약 1.9us 큰 이유는, 서버에서 메시지를 수신하기 전에 클라이언트와 연결된 소켓 디스크립터의 상태변화를 관찰하기 때문이다.



[그림 3] 응답시간

RPC/SOVIA는 RPC/TCP에 비하여, 메시지 크기가 작을 경우, 약 5.6배의 성능향상이 있고, 메시지 크기가 4K정도로 클 경우 약 4.7배의 성능향상이 있다. 메시지 크기가 커질수록 전체 시간에서 메시지 전송시간이 차지하는 비율이 작아지기 때문에 빠른 전송 매체를 사용하는 효과가 떨어지게 된다. 그러나 대부분 메시지 크기가 작으므로[3] 약 5배 이상 성능향상을 기대할 수 있다.

5. 결론

VIA는 일종의 인터페이스 규격으로 앞으로 하드웨어적으로 지원하는 VI NIC가 다수 출현할 것으로 전망된다. (예: Tandem Servernet II, Giganet cLAN 등) 이에 따라 클러스터 시스템을 위한 경량 고속 통신에 많이 활용될 전망이다. VIA는 성능 향상을 위해서 메시지 전송에 관련된 기본 기능만 제공하고 있으므로 그 외의 기능은 프로그래머가 부담해야 한다. 예를 들어, 흐름제어, 수신측의 디스크립터 준비와 같은 문제를 프로그래머가 신중히 고려하여야 한다.

그러한 환경에서 VIA의 성능을 사용할 수 있는 편리하고 안정적인 분산 프로그래밍 도구가 필요하다. 본 논문에서는 SOVIA를 전송 프로토콜로 사용하여 RPC를 구현한 RPC/VIA에 대해서 소개하였다. 기존의 TCP/IP 프로토콜을 사용하는 것보다 월등히 성능을 향상 시키면서 RPC 프로그래밍 환경은 그대로 유지하여 프로그램의 용이성과 고속네트워크 이용성을 동시에 실현하였다. RPC/VIA는 VIA를 사용할 때 안정적이고 편리한 분산 프로그래밍 환경을 제공한다. 또한 [7]을 바탕으로 최소한의 소켓 인터페이스를 구현하여 RPC/VIA 구현시 RPC 코드 수정을 최소화하였다.

참고문헌

[1] Rajkumar buyya, "High performance cluster computing," Prentice Hall, vol 1, 1999.

[2] Virtual Interface Architecture Specification, draft revision 1.0, 1997.

[3] Philip Buonadonna, Andrew Geweke, David Culler, "An Implementation and Analysis of the Virtual Interface Architecture," *Proceedings of SC'98*, Orlando, Florida, November 1998.

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

[5] Richard Stevens, "TCP/IP Illustrated," vol. 1, 1994.

[6] 김강호, 김진수, 김해진, "리눅스상에서 VIA 성능 비교," 한국정보과학회 가을 학술발표 논문집, 27권 2호, 2000.

[7] 김진수, 김강호, 김해진, "리눅스 클러스터 상에서 VIA를 이용한 효율적인 메시지 전달 프리미티브의 설계 및 구현," 병렬처리학회 논문집 11권 2호, 2000.

[8] Jin-Soo Kim, Kangho Kim, and Sung-In Jung, "Building a High Performance Communication Layer over Virtual Interface Architecture on Linux Clusters," 발표예정, 2000.

[9] Steven H. Rodrigues, Thomas E. Anderson, David E. Culler, "High-Performance Local Area Communication With Fast Sockets," USENIX, 1997.