MPICH-GP : 그리드 상에서 사설 IP 클러스터 지원을 위한 MPI 확장

박 금 례[†] · 윤 현 준[††] · 박 성 용[†††] · 권 오 영[††††] · 권 오 경[†††††]

요 약

그리드 네트워크에서 MPI를 사용하여 지리적으로 산재된 컴퓨팅 자원을 활용하여 복잡한 문제를 해결하기 위한 MPICH-G2는 사설 IP 클러스터를 지원하지 못한다는 단점을 가지고 있다. MPICH-G2의 경우에는 모든 계산노드가 외부에 노출되어 악의적인 보안 침해의 가능성이 높아지게 된다. MPICH-GP는 NAT와 프락시를 병용하여 사설 IP 클러스터의 문제를 해결한다. 사설 IP 클러스터의 프론트 노드에 프락시를 두고, 이를 통해 내부 계산 노드로의 통신 링크를 중계한다. 따라서 사설 IP 클러스터와 공인 IP 클러스터가 혼재된 네트워크에서도 적정한 경로를 설정하고 성공적으로 MPI 작업을 수행할 수 있다. 본 논문에서는 MPICH-GP의 성능을 기존의 MPICH-G2와 비교하였다. 그리드 환경에서 MPICH-GP는 MPICH-G2의 80% 이상의 성능을 보이며, RANK 관리기법을 적용한 경우는 95% 이상의 성능을 나타낸다.

키워드 : 그리드, MPI, MPICH-G2, 사설IP클러스터

MPICH-GP : An MPI Extension to Supporting Private IP Clusters in Grid Environments

Kumrye, Park[†] · Hyunjun, Yun[††] · Sungyong, Park[†††] · Ohyoung, Kwon[††††] · Ohkyoung, Kwon[†††††]

ABSTRACT

MPICH-G2 is an MPI implementation to solve complex computational problems by utilizing geographically dispersed computing resources in grid environments. However, the computation nodes in MPICH-G2 are exposed to the external network due to the lack of supporting the private IP clusters, which raises the possibility of malicious security attacks. In order to address this problem, we propose MPICH-GP with a new relay scheme combining NAT(Network Address Translation) service and an user-level proxy. The proxy running on the front-end system of private IP clusters forwards the incoming connection requests to the systems inside the clusters. The outgoing connection requests out of the cluster are forwarded through the NAT service on the front-end system. Through the connection path between the pair of processes, the requested MPI jobs can be successfully executed in grid environments with various clusters including private IP clusters. By simulations, we show that the performance of MPICH-GP reaches over 80% of the performance of MPICH-G2, and over 95% in case of using RANK management method.

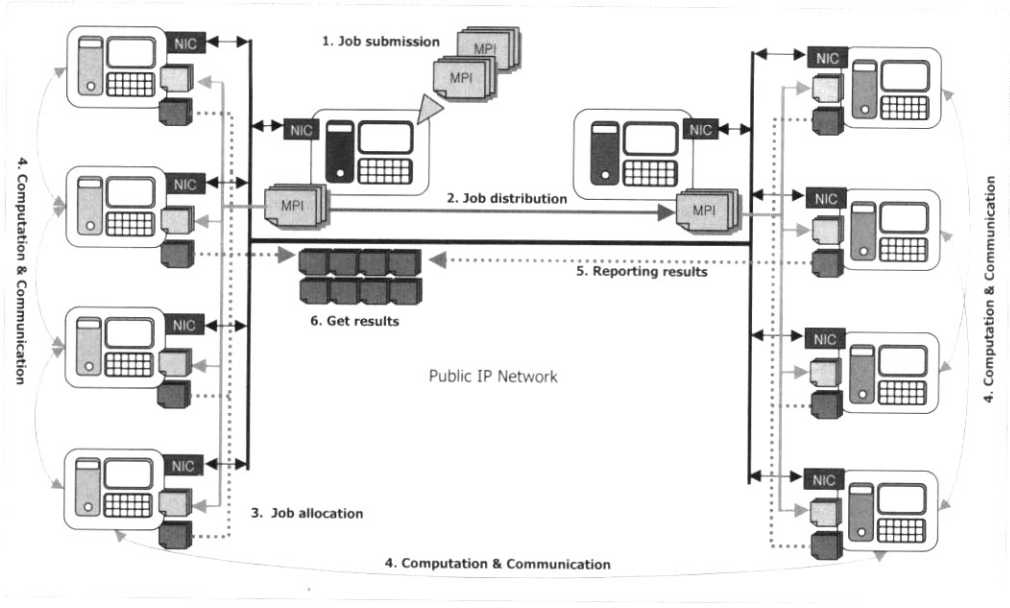
Key Words : Grid, MPI, MPICH-G2, Private IP Cluster

1. 서 론

기존의 분산 및 병렬 컴퓨팅 환경은 이제 하나의 조직 내의 슈퍼컴퓨터를 벗어나 지역적으로나 조직적으로 분리되어 있는 슈퍼컴퓨터, 대용량 저장장치, 다양한 고성능 연구 장비와 데이터를 공유하는 그리드 컴퓨팅(Grid Computing)[1] 환경으로 급속히 진보해 나가고 있다. 기능적으로 그리드는 다양한 분산된 이기종의 컴퓨팅 자원과 데이터 자원을 단일한 형태로 보고 접근할 수 있도록 하는 툴이자 미들웨어이고 서비스로서 광범위하게 분산된 응용 시스템의 구축, 관리 및 사용을 지원한다.

이러한 그리드 컴퓨팅 환경을 구축하기 위해서 표준적으로 사용되는 것이 글로버스 툴킷(Globus Toolkit)[2]이다. MPI(Message Passing Interface)[3]는 이러한 그리드 자원이 가지는 컴퓨팅 파워를 활용하여 한대의 서버에서 해결할 수 없는 복잡한 문제들을 해결하기 위해 사용된다. 초기의 MPI는 여러 병렬 컴퓨터에서 효율적으로 수행되는 응용 프로그램을 작성하기 위해 개발되었으며, MPI 구현 시스템으

※ 본 연구는 한국과학기술정보연구원(KISTI)의 지원으로 수행되었음.
† 정 회 원 : 삼성전자 정보통신총괄 무선사업부 WiBro S/W Lab
†† 준 회 원 : 서강대학교 컴퓨터학과 석사과정
††† 정 회 원 : 서강대학교 컴퓨터학과 조교수/부교수
†††† 종신회원 : 한국기술교육대학교 부교수
††††† 정 회 원 : KISTI 슈퍼컴퓨팅센터 연구원
논문접수 : 2006년 7월 28일, 심사완료 : 2006년 12월 6일



(그림 1) 그리드 환경에서의 MPI 수행과정 및 문제점

로 가장 널리 사용되고 있는 것이 미국 ANL(Argonne National Laboratory)에서 구현한 MPICH[4]이다. 이를 그리드 환경에 적용한 것으로 MPICH-G[5], MPICH-G2[6] 등이 있으며, MPICH-G2가 사실상의 표준으로 사용되고 있다.

그리드 환경은 지역적으로, 그리고 조직상으로 분리된 여러 자원을 하나로 통합한다. 자연스럽게 대부분의 그리드에는 하나 이상의 클러스터들이 존재하게 된다. 글로버스를 이용하여 구축된 (그림 1)의 그리드가 있다. 각 계산 노드들은 할당된 작업을 수행하는 동안 다른 계산 노드들과 직접 연결을 맺어 통신을 하게 되는데, 전형적인 클러스터 설정인 사설 IP 주소로 구성되어 있다면, 요청된 작업을 성공적으로 끝마칠 수 없다. 그렇기 때문에 현재 널리 이용되고 있는 그리드 상에서의 MPI 구현인 MPICH-G2[6][7]는 계산에 참여하는 모든 노드가 공인 IP 주소를 가지도록 요구한다. 이러한 설정은 그리드 환경에서 MPI 작업의 수행을 가능케 하였지만, 기존 클러스터 설정이 가지는 장점을 훼손함으로써 다음의 문제를 야기하였다.

① 모든 계산노드에 공인 IP 주소가 할당됨에 따라, 계산노드가 외부에 노출되므로 보안 침해의 소지가 높아지고, 이에 따라 클러스터의 관리가 복잡해진다.

② (그림 1)의 그리드 환경에서는 총 10개의 공인 IP 주소가 필요하지만, 사설 IP 클러스터로 구성하는 경우, 단지 2개의 IP 주소로 충분하다. 실제 클러스터들은 보통 8개 이상의 계산노드로 구성되므로, 사설 IP 클러스터를 그리드에 적용함에 따라 얻을 수 있는 IP 주소 부족의 해결은 상당한 것이다.

기존의 그리드 환경을 위한 MPI가 하부의 클러스터 구조를 고려하지 않고 MPI를 그리드 상에서 수행만 가능하도록 일률적으로 공인 IP 주소를 가지도록 했다. 이러한 단점을 해결하여 사설 IP 주소로 구성된 클러스터, 사설 IP 클러스터와 공인 IP 클러스터가 혼합되어 구성된 그리드 환경에서도 글로버스를 통한 MPI 작업을 수행할 수 있도록 해야 한다.

본 논문은 MPICH-G2를 수정하여 사설 IP 클러스터 노드와의 통신을 중계하기 위해서 NAT와 유저 영역에서 동작하는 프락시를 결합한 방법을 설계하여 적용하여 MPICH-GP(Grid-enabled MPI supporting the Private IP cluster)를 구현하였다.

서로 다른 클러스터에 존재하는 사설 IP 노드간의 통신을 연결하기 위해서는 (사설 IP 주소 계산노드)-(공인 IP 주소 프론트 노드)-(공인 IP 주소 프론트 노드)-(사설 IP 주소 계산노드)를 거쳐야 한다. 이와 같은 상황에서 통신을 중계하기 위해 고려할 수 있는 방법은 구현 영역, 프론트 노드간의 연결의 수, 차별화된 내부 통신 라이브러리 등에 따라 크게 NAT[7]와 게이트웨이(Gateway), 프락시 등으로 나눠볼 수 있다. 성능에 가장 큰 영향을 끼치는 것은 구현 영역으로 커널 영역에서 구현된 NAT와 게이트웨이가 유저 영역에서 구현된 프락시에 비해 두 배 정도 높은 통신 대역폭을 보인다[8].

공인 IP 주소로 구성되는 MPICH-G2에서는 계산에 참여하는 노드들 간에 통신하기 위해서 호스트명이나 IP 주소를 가지고 직접 접근이 가능하다. 그러나 사설 IP 클러스터의 경우, 목적지 사설 IP 주소 계산노드의 위치를 파악할 수 있는 방법이 없다. 그러므로 MPI 초기화 과정에서 생성되는 프로세스들의 정보 테이블을 작성하는 시점에서 추가적인 정보가 제공되어야 한다. MPICH-GP에서는 GP_GUID(Globally Unique ID) 개념을 도입하여 한 쌍의 노드들이 공인 IP 클러스터인지 사설 IP 클러스터인지를 구분하고 이에 따라서 목적지 노드가 공인 IP 주소를 가지고 있는 경우는 NAT를

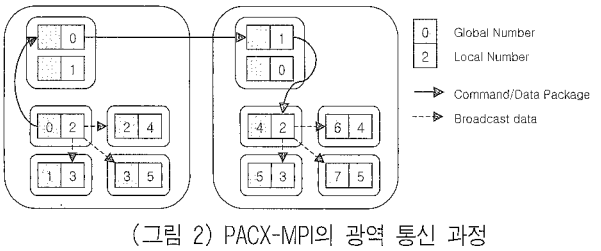
통해서 바로 연결을 맺고, 사설 IP 주소를 가지고 있는 경우는 상대 프론트 노드의 프락시 데몬을 거쳐서 연결을 맺는 방식으로 통신을 시작하게 된다. 프락시는 특정 포트 번호에서 클라이언트의 연결 요청을 기다리고 있다가 연결 요청이 있으면 받아들이고 클라이언트로부터 목적지 주소와 포트 번호 등의 정보를 받아서 해당 목적지로 연결을 요청한다. 연결이 성립되면 연결을 요청한 노드로 연결이 맺어졌음을 알리고 데이터가 들어오면 다른 쪽으로 중계한다.

이후의 본 논문은 다음과 같이 구성된다. 2장에서는 1장에서 제시한 문제를 해결하는데 적용할 수 있는 기존의 방안들에 대한 연구를 토대로 사설 IP 주소 문제를 해결하기 위한 방법을 모색하고 기존 연구의 한계점을 밝힌다. 3장에서는 2장의 연구들을 토대로 사설 IP 클러스터의 통신을 중계하기 위하여 NAT와 프락시를 결합한 방법을 채택하여 MPICH-GP를 구현하며, MPICH-GP의 중요 구성요소들에 대해서 설명한다. 4장에서는 LAN과 WAN 환경에서 MPICH-GP와 MPICH-G2의 성능을 비교 측정하여 프락시에서의 통신 중계에 따른 성능 오버헤드를 알아본다. 응용 프로그램의 NPB 벤치마크[16]를 통하여 실제 응용 프로그램에서 프락시에 걸리는 지연시간의 정도를 측정하며, 노드 수를 변화시켜 가면서 프락시의 확장성을 테스트한다. 마지막으로 5장에서는 논문의 결론과 향후 과제에 대해서 논하도록 한다.

2. 관련 연구

2.1 PACX-MPI

PACX-MPI(PArallel Computer eXtension)[11]은 서로 다른 MPP(Massively Parallel Processing System, 즉, 클러스터)들을 하나로 통합하려는 시도에서 출발한 라이브러리이다. 이 라이브러리를 사용하여 컴파일하면 기존에 작성된 응용 프로그램의 코드를 수정하지 않고도 다수의 클러스터에서 마치 한 클러스터를 사용하는 것처럼 MPI 작업을 수행할 수 있다. PACX-MPI는 클러스터 내부의 통신을 위해서는 벤더가 제공하는 MPI를 사용하여 빠르게 통신하고, 클러스터 간의 통신에는 TCP나 UDP등의 표준적인 프로토콜을 사용한다. 클러스터간의 통신을 위해서 PACX-MPI는 프론트 노드에 통신 프로세스를 따로 두고 있다. 이들 프로세스는 직접 계산에 참여하는 것이 아니라 클러스터간의 통신을 중계하기 위한 프락시 서버 역할을 하게 된다. 한 프로세스는 데이터 송신을 위한 노드, 다른 한 프로세스는 데이터 수신을 위한 노드이다.



(그림 2) PACX-MPI의 광역 통신 과정

(그림 2)는 PACX-MPI에서 집합 연산의 수행과정을 보여준다. 광역번호 0인 노드가 브로드캐스트 연산을 수행하는 경우 내부 클러스터들로는 직접 브로드캐스트를 하고, 외부 클러스터에 대해서는 송신 노드를 통해서 메시지를 전달, 각 클러스터의 수신 노드는 이 메시지를 받아서 자신의 노드 중에서 가장 낮은 지역번호를 가진 노드에게 메시지를 전달하며, 전달받은 노드는 클러스터 내부의 모든 노드에게 메시지를 브로드캐스트 한다.

2.2 방화벽 환경을 지원하는 MPICH-G

방화벽이 설치되어 있는 환경에서 MPICH-G[5]를 사용하기 위하여, 방화벽을 넘어선 통신 링크를 맺기 위한 RMF(Resource Manager beyond the Firewall)라 부르는 자원관리자와 방화벽을 넘어 TCP 통신 링크를 중계하기 위한 Nexus[12] Proxy가 존재한다.

방화벽으로 가려져 있는 노드들의 통신을 중계하기 위해서 Nexus Proxy는 방화벽 안팎의 TCP 연결을 중계한다. Nexus Proxy의 외부서버는 방화벽 바깥에서 실행되고 Nexus 내부 서버는 방화벽 안쪽에서 데몬 프로세스로 실행된다. 방화벽 안쪽에서 실행 중인 프로세스가 외부 호스트로 연결을 시도하거나 소켓을 바인드 할 때, 클라이언트 프로세스는 직접 *connect()*나 *bind()*를 호출하여 포트 번호에 연결하거나 바인드 하는 대신에 Nexus Proxy의 외부 서버로 중계 요청을 보낸다.

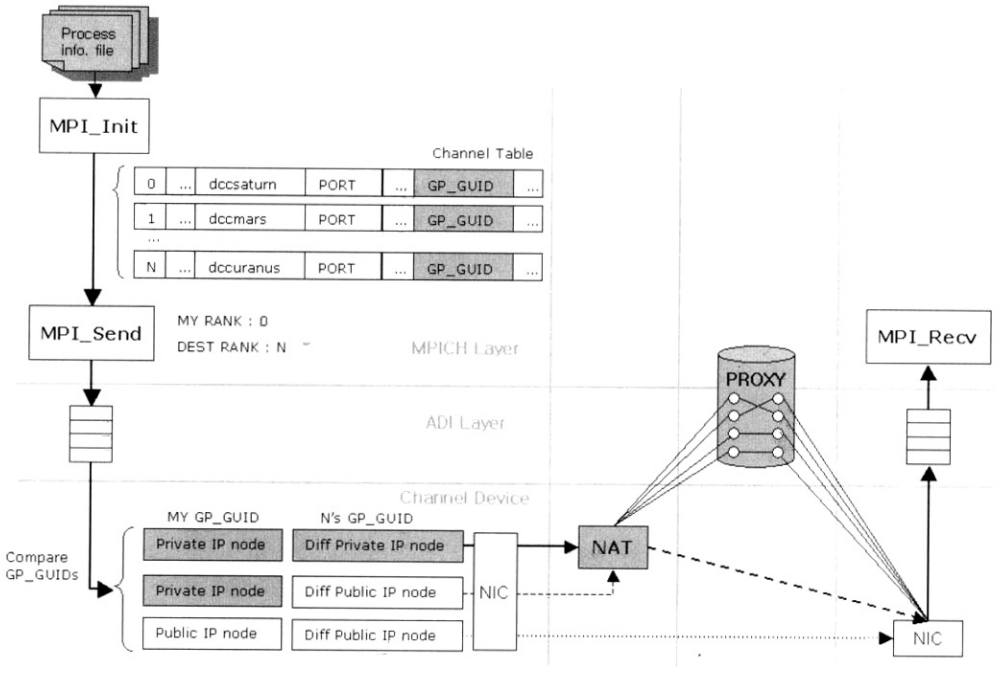
2.3 기존 연구들의 문제점

PACX-MPI는 프론트 노드에 송/수신 노드를 두고, 이를 통해서 메시지를 중계하는 방식을 취하고 있는데 이것은 일반적인 유저 영역에서 구현된 프락시로 볼 수 있다. 유저 영역의 프락시를 통한 통신 중계 방법을 그대로 도입하여 사설 IP 클러스터의 문제를 해결하고자 하는 경우 유저 영역과 커널 영역으로의 문맥전환이 두 번이나 일어나게 되고 이에 따라 직접 연결을 맺는 경우와 비교하여 상당한 성능저하가 예상된다.

방화벽 환경을 지원하는 MPICH-G의 경우는 방화벽을 통과하여 연결을 맺기 위해서 여러 단계의 과정을 거치게 된다. 연결 맺는 과정에서의 지연시간도 클 것이며, 실제 연결 맺는 시간을 고려하지 않는다고 할지라도 중간에 연결을 맺기 위해 통과한 서버들을 차례로 경유하여 통신 메시지를 전달하게 된다. Nexus Proxy의 외부/내부 서버들은 유저 영역에서 구현되었으므로 각각을 지남에 따르는 중계 오버헤드는 오히려 PACX-MPI가 가지는 오버헤드보다 크게 나타나는 단점이 있다. 방화벽 환경을 지원하는 MPICH-G의 경우, 논문을 통해서 발표는 되었으나 실제로 배포되어 사용되고 있는 것은 아니다.

3. MPICH-GP : 그리드 상에서 사설 IP 클러스터 지원을 위한 MPI 확장

MPICH-GP는 공인 IP 클러스터만으로 구성된 그리드를



(그림 3) MPICH-GP의 구조

위한 MPI 구현인 MPICH-G2를 확장하여 사설 IP 클러스터로 구성된 그리드, 더 나아가 공인 IP 클러스터와 사설 IP 클러스터가 혼재하는 그리드 환경까지도 지원한다. 앞의 'G'는 Grid를, 뒤의 'P'는 Private IP를 의미한다.

3.1 MPICH-GP 구조

MPICH-GP는 (그림 3)과 같은 구조를 갖는다. 초기화 단계와 실제 통신 단계로 나눠볼 수 있는데 프로세스간의 통신 단계에서는 경로 결정 단계와 경로에 따라서 실제 연결을 맺고 메시지를 전송하는 과정을 거치게 된다.

MPICH-GP의 초기화 단계에서는 파일을 통해 제공되는 참여 노드들의 정보를 통해 채널 테이블을 구성하게 된다. 이 과정에서 사설 IP 클러스터를 지원하기 위한 장치로서 사설 IP 주소를 포함한 모든 노드들에 대해 고유한 ID인 GP_GUID를 생성하여 채널 테이블 정보에 포함하게 된다. GP_GUID는 프론트 노드명과 계산 노드명, 그리고 프로세스 ID로 구성되는데, 이 정보를 통해 클러스터 내부에서만 유효한 사설 IP 주소의 경우에도 각 노드를 구별하고 실제 통신할 수 있게 된다.

MPICH-GP의 통신단계에서는 GP_GUID를 비교하여 연결 경로를 설정하는 과정과 이에 따라 연결을 맺고 실제 메시지를 전송하는 단계로 나눠볼 수 있다. 두 노드간의 통신 연결을 맺기 직전에 초기화 과정에서 구성한 채널 테이블의 GP_GUID를 비교하는 과정을 거친다. 이를 통해서 나와 상대 노드가 사설 IP 주소를 가지는 노드인지 공인 IP 주소를 가지는 노드인지를 비교하고 이에 따라 연결을 맺게 된다. 내가 사설 IP 주소를 가진 노드인 경우는 프론트 노드에서 제공하는 NAT 서비스를 통해서 외부로 나아가게 되는데, 상대 노드가 사설 IP 주소를 가지고 있다면 상대 노드로 직접적인 통신 링크를 생성하는 것은 불가능하다. MPICH-GP에서는 클러스터 내부에서 외부로 나가는 연결을 포워드 해주는 NAT 서비스를 프론트 노드에서 제공하며, 프론트 노드에 유저 영역에서 구현된 프락시를 두어서 내부 노드로의 통신 링크를 중계한다.

3.1.1 MPICH-GP 수행 과정

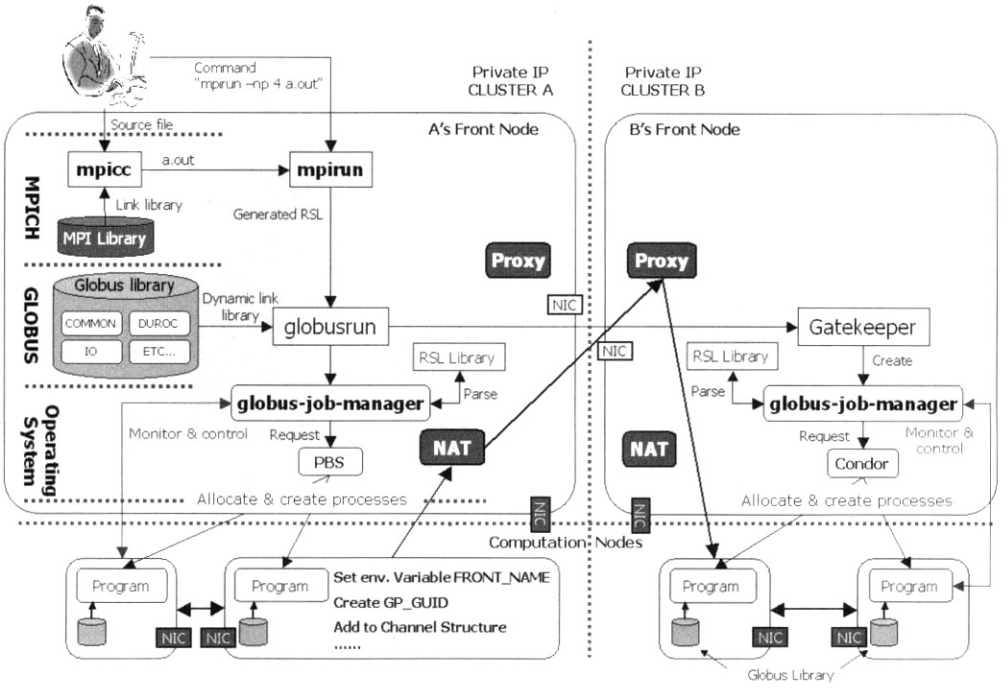
(그림 4)는 두개의 사설 IP 클러스터로 구성된 그리드 네트워크에서 MPICH-GP의 수행과정을 보여준다. 그리드 네트워크를 구성하기 위한 미들웨어로서 글로버스 툴킷을 사용하고 있다. MPI의 라이브러리는 실행파일에 링크되어 전달되지만, 글로버스 라이브러리는 실행 시에 동적으로 링크되기 때문에 계산 노드들에게는 글로버스 라이브러리가 제공되어야 한다.

작성된 사용자의 MPI 프로그램은 mpicc를 통하여 컴파일하고, mpirun을 통해 RSL 파일을 생성하여 글로버스의 자원관리서비스를 담당하는 GRAM의 클라이언트인 globusrun에게 전달된다.

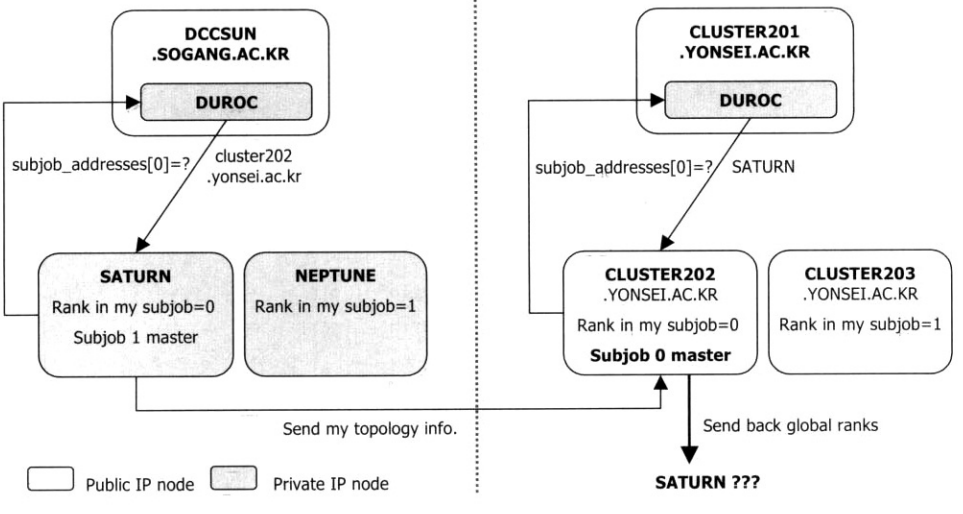
① globusrun은 작업을 전달해주는 역할을 하는데, 자신의 클러스터에서는 직접 로컬 작업 관리자(globus-job-manager)에게 전달하고, 외부 클러스터인 경우는 글로버스의 gatekeeper를 통하여 로컬 작업 관리자를 생성하여 전달한다.

② 로컬 작업 관리자는 전달받은 작업을 계산노드에게 할당하여 작업을 수행하게 된다.

③ 계산노드에 생성된 MPI 프로세스들은 초기화 과정을



(그림 4) MPICH-GP의 수행과정



(그림 5) 사설 IP 클러스터에서 DUROC의 문제점

수행한다. 여기서 초기화 과정이란 MPI 프로세스를 구동하기 위해 필수적인 정보들을 담고 있는 채널 테이블을 구성하는 과정이다. 각 프로세스들의 호스트명, 통신 포트 번호, LAN_ID, 그리고 WAN_ID 등의 정보, 전체 프로세스 가운데 자신의 위치를 나타내는 랭크(RANK) 정보가 요구된다. MPICH-G2에서는 글로버스의 DUROC(Dynamically-Updated Request Online Coallocator)[13]과의 통신을 통해서 이들 정보를 알아오게 되지만, 사설 IP 클러스터의 경우 DUROC과의 통신이 불가능하기 때문에 MPICH-GP에서는 파일을 기반으로 이들 정보를 제공한다.

④ MPICH-GP에서는 ③에서 언급한 정보에 추가적으로 사설 IP 클러스터를 구분하고 위치를 파악하여 통신을 가능케 하기 위해 사용되는 GP_GUID(globally unique identifier)를 생성하여 채널 테이블에 포함한다. GP_GUID는 프론트 노드명, 계산 노드명, 프로세스 ID로 이루어지는 전역 ID이다.

⑤ 초기화 과정이 끝나면 각 프로세스는 자신에게 할당된 작업 수행을 시작한다. 이 과정 중에 다른 프로세스와 통신을 하게 되는데 초기화 과정에서 생성한 채널 테이블의 GP_GUID를 비교하여 상대노드가 사설 IP를 가지고 있을 경우는 상대 프론트 노드에서 실행

중인 프락시의 특정 포트(이미 알려진 포트) 번호로 접속하여, 최초의 목적지의 정보를 보내고 연결을 요청한다.

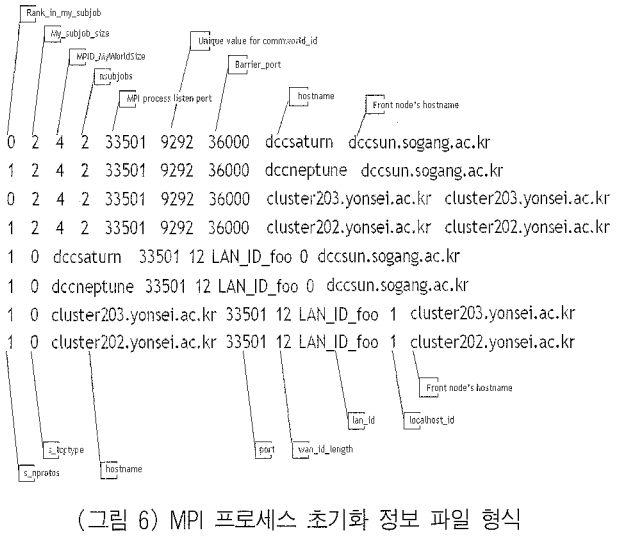
⑥ 모든 노드에서 작업이 완료되면 결과는 사용자에게로 출력된다.

3.1.2 MPICH-GP 초기화 과정

MPICH-GP에서 사설 IP 클러스터를 지원하기 위해서 초기화 과정에 큰 변화가 있다. MPICH-G2에서는 글로버스의 DUROC을 이용하여 초기화를 진행하는데, DUROC은 여러 로컬 작업관리자에게로 작업을 분배하고, 작업에 참여하는 로컬 작업 관리자에게 다른 로컬 작업관리자의 주소와 작업의 크기 등의 기초적인 정보를 제공한다. MPICH-G2는 초기화 과정에서 DUROC이 제공하는 동적인 정보를 통해서 토폴로지를 구성하고 서브잡(subjob) 단위로 자신들의 프로토콜 정보를 교환하여 전역 랭크(global rank)를 계산하고, 이에 따라서 채널 테이블을 구성한다.

사설 IP 클러스터의 경우는 (그림 5)에서 보는 것과 같이 DUROC에게 다른 서브잡 마스터와 통신하고자 하지만, DUROC이 제공한 주소는 사설 IP 이므로 외부 클러스터의 노드들이 접근할 수 없다. 결과적으로 토폴로지를 생성하는 단계에서 다른 서브잡 마스터들과의 정보교환이 이루어지지 않기 때문에 MPI 작업을 제대로 수행될 수가 없다. 이러한 문제로 인해서 MPICH-GP에서는 DUROC 모듈을 걷어내고, 클러스터 관리자가 작성한 파일에서 필요한 정보를 읽어 와서 채널 테이블을 구성하게 된다. 각 프로세스의 listen port 번호와 같이 동적인 환경에서 결정되는 정보도 존재하게 되는데, 이러한 정보 역시 정적으로 할당된다.

(그림 6)은 *dccsun*을 프론트 노드로 하며 *dccsaturn, dccneptune*을 계산 노드로 갖는 클러스터와, *cluster201*을 프론트 노드로 갖는 *cluster203, cluster204*의 계산 노드에서 두 개의 서브잡을 구동하는 환경에 대한 프로세스 정보 파일이다. 파일의 각 필드와 그 의미는 <표 1>에 나타내었다.



(그림 6) MPI 프로세스 초기화 정보 파일 형식

〈표 1〉 프로세스 정보 파일의 각 필드와 역할

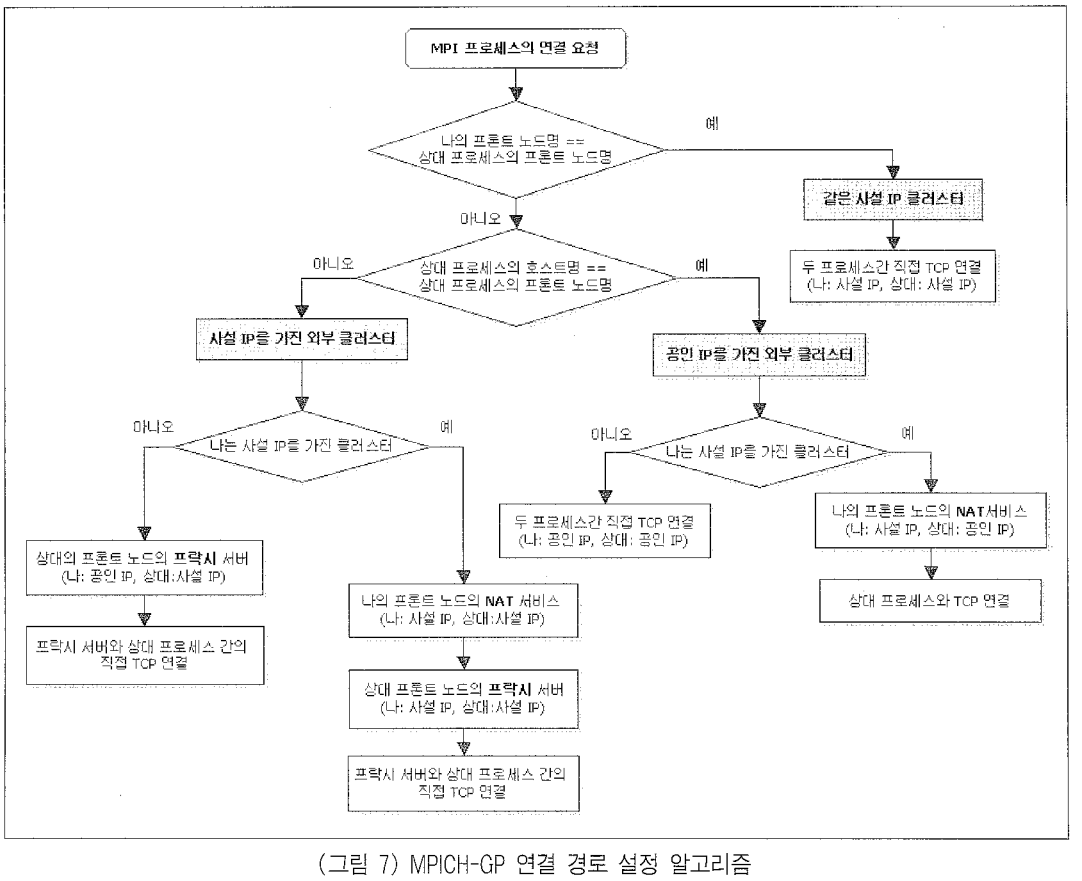
필드	역할
rank_in_my_subjob	자신이 속해있는 subjob내의 랭크
my_subjob_size	자신이 속해있는 subjob의 크기
MPID_MyWorldSize	전체 MPI job의 크기
nsubjobs	subjob의 수
MPID_MyWorldRank	전체 MPI job에서의 자신의 랭크
MPI process listen port	MPI 프로세스의 listen port 번호
Unique value for commworld_id	commworld_id를 만들기 위한 추가적인 값, 임의의 unique값을 할당한다.
s_nprotos	사용되는 프로토콜의 수
s_tcptype	사용되는 프로토콜, enum proto {tcp, mpi, unknown};
hostname	프로세스가 수행될 노드의 호스트명
front node's hostname	프로세스가 수행되는 노드의 프론트 노드명
port	노드의 MPI process의 listen port 번호
wan_id_lng	wan_id의 길이
wan_id	사용자가 지정하는 계층별 전송 weight
lan_id_lng	lan_id의 길이
lan_id	사용자가 지정하는 계층별 전송 weight
localhost_id	localhost-TCP와 LAN/WAN-TCP를 구별

3.2 사설 IP 클러스터의 통신 중계 방법

사설 IP 클러스터를 지원한다는 것은 바로 사설 IP 주소 노드로의 통신을 중계한다는 것이다. 사설 IP 주소는 클러스터 내부에서만 유효하기 때문에 외부 네트워크와의 통신을 중계하기 위해서는 외부 네트워크에서 인식가능한 공인 IP 주소를 가지고 있는 프론트 노드에서 통신을 중계해야 한다.

통신을 중계하기 위한 방법으로 크게 게이트웨이와 같은 커널 수준에서의 중계 방법과 흔히 프락시로 알려진 유저 영역에서의 중계 방법을 생각해볼 수 있다. 유저 영역의 경우는 일단 네트워크를 통해서 전달된 데이터를 유저 영역의 버퍼로 복사하여 해당 작업을 수행하고 다시 커널 영역의 버퍼에 데이터를 쓰게 되고, 실제 네트워크로 데이터를 전송하기 위해서 네트워크 디바이스의 버퍼로 복사되는 과정을 거치게 된다. 커널 수준에서의 구현은 커널 영역에서 직접적으로 처리되어 네트워크로 전송되기 때문에 유저 영역으로의 복사나, 유저 영역에서 다시 커널 영역으로의 복사 과정과 같은 문맥 전환 비용이 없기 때문에 훨씬 빠르게 처리할 수 있다. 그러나 커널 영역에서 구현하기 위해서는 커널 내부의 함수들을 이용하여 통신을 중계하기 위한 장치를 구현, 커널을 컴파일 해줘야 한다. 이러한 과정은 프로그램의 이식성을 떨어뜨린다.

전통적인 클러스터의 설정을 살펴보면 사설 IP 클러스터의 경우 프론트 노드에서 NAT 서비스 제공하여 외부 네트워크로 연결할 수 있도록 한다. NAT 서비스는 커널 영역에서 구현된 것으로 리눅스에서 기본적으로 제공된다. 그러나 이 경우에는 외부에서 내부로는 연결할 수 없다. MPICH-GP에서는 외부로 나가는 연결은 NAT 서비스를 통해서 연결을 포워드하며 내부 노드로 들어오는 연결에 대해서는 프론트 노드에 프락시를 둠으로써 외부의 연결을 내부로 중계하는 방안을 채택하였다. NAT와 프락시를 병용한 중계 방안[14]은 커널 영역과 유저 영역의 중간적인 방법으로써 프락



(그림 7) MPICH-GP 연결 경로 설정 알고리즘

시를 통하여 외부로 나가는 연결과 내부로 들어오는 연결을 모두 처리하는 방안과 비교해서 성능이 우수하며, 커널 영역의 구현이 가지는 이식성의 문제를 해결할 수 있다.

3.3 전역 ID 생성 관리

일반적으로 MPI 프로세스는 자신의 위치를 말해주는 랭크를 가지고 채널 테이블에서 해당 채널 정보를 참조하여 통신을 진행한다. 그러나 사설 IP 클러스터의 경우, 호스트명만으로는 정확한 위치를 알아낼 수가 없기 때문에 문제가 된다.

MPICH-GP에서는 사설 IP 클러스터 각각에 고유한 ID인 GP_GUID를 생성하여 노드들의 통신을 관리함으로써 이 문제를 해결한다. GP_GUID는 프론트 노드의 호스트명, 계산 노드의 호스트명과 프로세스 ID로 구성된다. 각 노드들은 자신의 GP_GUID를 가지고 상대 노드의 GP_GUID를 비교함으로써, 상대 노드가 속해있는 클러스터의 위치와 클러스터의 종류를 알아내고 그에 따라 연결 경로를 달리하여 연결을 맺게 된다. FRONT_NAME을 비교함으로써 상대 노드가 같은 클러스터 내에 있는지를 파악할 수 있으며, 같은 클러스터인 경우 직접 연결을 맺을 수 있다. FRONT_NAME은 환경 변수를 통해서 얻어오는데, 사설 IP 클러스터의 경우 실제 프론트 노드명을 값으로 설정하면 된다. 공인 IP 클러스터인 경우 프론트 노드를 통한 통신 중계가 필요치 않으므로, FRONT_NAME을 자신의 노드명과 동일하게 설정한다. 이와 같은 설정을 통해서 상대 노드가 사설 IP 클러스터인지 공인 IP 클러스터인지를 구분할 수 있다. 사설 IP 클러스터인 경우는 FRONT_NAME과 COMPUTE_NAME이 다르며 공인 IP 클러스터인 경우 이 둘이 같은 값을 갖는다.

MPICH-GP에서는 초기화 과정에서 GP_GUID를 먼저 생성하여 채널 테이블에 포함시킨다. 이로써 GP_GUID는 프로세스가 종료될 때까지 유지되는 채널 테이블의 중요한 하나의 속성으로 자리 잡게 된다.

(그림 7)은 각 프로세스가 자신의 GP_GUID와 상대 노드의 GP_GUID를 비교하여 통신 경로를 찾아가는 과정을 도시한 간단한 알고리즘이다.

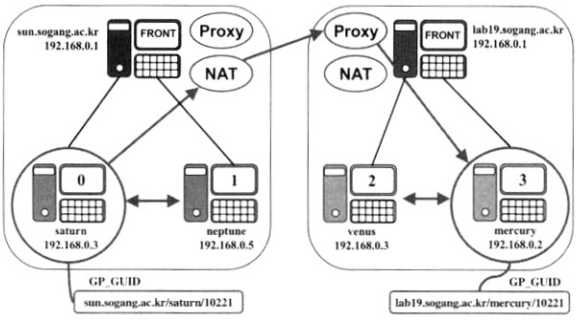
3.3.1 두 개의 사설 IP 클러스터에서의 통신

(그림 8)은 두개의 사설 IP 클러스터 상에서 MPICH-GP에서의 통신과정을 보여준다. *saturn*과 *neptune*은 같은 프론트 노드이므로, 같은 클러스터에 존재한다고 판단, 사설 IP 주소를 가지고 직접 연결을 맺고 통신을 한다. *saturn*과 *mercury*는 프론트 노드가 다르므로 *mercury*의 프론트 노드에서 대기하고 있는 프락시로 접속한다. 프락시 포트 번호는 이미 알려져 있으며, 프락시는 여러 클라이언트로부터 접속을 받아들인다. *mercury*의 프론트 노드인 *lab19.sogang.ac.kr*로 연결을 맺은 후에, 목적지 노드인 *mercury*의 GP_GUID를 프락시에게 전송한다. 프락시는 이 정보를 가지고 목적지 *mercury*로 연결을 맺고, 양쪽 연결이 모두 성립되면, 이

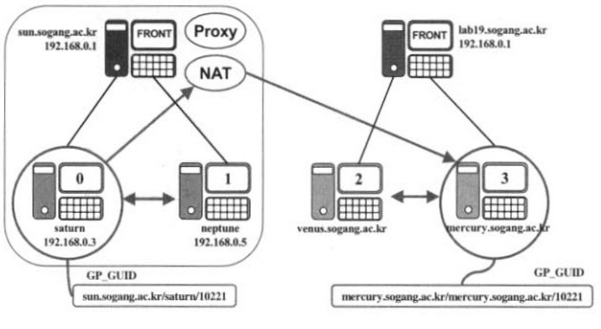
후부터 프락시는 이들 연결 사이에서 메시지를 중계하는 역할을 한다.

3.3.2 사설 IP 클러스터와 공인 IP 클러스터가 혼재된 환경에서의 통신

공인 IP 클러스터로 구성되어 있는 기존 MPICH-G2 환경이나 공인 IP 클러스터와 사설 IP 클러스터가 혼합된 환경에서의 통신 방법은 사설 IP 클러스터만으로 구성된 환경에서의 통신 방법과는 조금 달라져야 한다. 물론 위와 같은 방법을 통해서도 MPI 프로그램을 수행할 수 있다. 이 경우는 공인 IP 주소를 가진 노드에 직접(프락시를 거치지 않고) 연결을 맺는 것과 비교하면 프락시를 통과하는 오버헤드가 크기 때문에 현저한 성능 저하를 야기한다. 그러므로 공인 IP 클러스터와 사설 IP 클러스터를 구별하여 각각의 경우, 다른 통신 경로를 선택해서 연결을 맺어야 한다.



(그림 8) 두 개의 사설 IP 클러스터에서의 MPICH-GP 통신경로



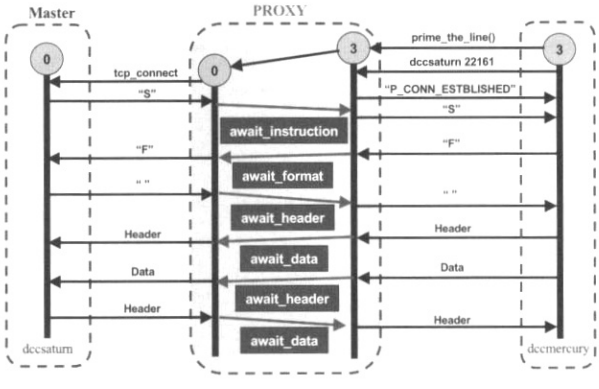
(그림 9) 사설 IP와 공인 IP 클러스터에서 MPICH-GP 통신경로

(그림 9)는 사설 IP 클러스터와 공인 IP 클러스터가 혼합된 환경에서 MPICH-GP의 통신과정을 보여준다. 사설 IP 클러스터에 있는 *saturn*이 공인 IP 클러스터에 있는 *mercury*와 통신하려고 한다. (그림 7)의 알고리즘에 따라, 맨 처음 *saturn*은 목적지 노드인 *mercury*와 프론트 노드 이름을 비교하고, 이것이 다르므로 *mercury*가 다른 클러스터에 있음을 인식한다. 두 번째로 *mercury*의 프론트 노드와 계산노드를 비교한다. 그림에서 살펴보듯이 *mercury*는 공인 IP 주소를 가지고 있으므로 프론트 노드명과 계산노드명이 동일하고, 이 경우에 *saturn*은 프론트 노드의 NAT 서비스를 거쳐서 바로 *mercury*로 연결을 맺는다.

이와 같은 방법은 NAT 서비스를 통해서 상대방 노드로 직접 연결하기 때문에 프락시로 연결을 맺는 시간과 커널과 유저영역 사이의 문맥전환에 따르는 오버헤드를 줄일 수 있다.

3.4 통신 프로토콜

MPICH-GP는 기존 MPICH-G2에서 사용하는 프로토콜을 그대로 사용한다. 통신하려는 두 노드간의 통신을 프락시를 통해서 중계하려면, 프락시 역시도 이들의 프로토콜을 이해하고, 이에 따라 구현되어야 한다.



(그림 10) 프락시를 통한 통신 중계 과정

(그림 10)은 통신 중계를 위한 데몬이 수행되는 과정을 그림으로 표현한 것이다. 프락시가 SRC와 DEST 사이의 연결을 중계하여 MPICH-GP의 프로토콜에 따라서 간단한 *MPI_Send/MPI_Recv*를 처리하는 과정을 보여준다.

① 프락시는 데몬의 형태로 실행되며 사전에 알려진 포트 번호에서 연결 요청을 기다린다.
② 프로세스 3은 프락시의 포트 번호로 접속을 하고 프락시에게 자신이 원하는 프로세스 0의 주소와 포트 번호를 보낸다.
③ 프락시는 전달받은 목적지 노드의 주소와 포트 번호를 토대로 프로세스 0과 연결을 시도한다.
④ 연결시도가 성공적으로 완료될 경우 프로세스 3과 맺은 연결을 통해 ACK("P_CONN_ESTABLISHED")를 보낸다. 이 과정이 완료되면 SRC-PROXY-DEST 간에 연결이 맺어진 상태가 된다.
⑤ MPI의 초기화가 끝나면, 각 계산노드에서 실제 MPI의 메인 부분을 실행하기 전에 각 프로세스의 시작시점을 맞춰주기 위한 동기화 루틴이 필요하다. 그림에서 "S"는 이때 보내지는 메시지이다.
⑥ 위와 같은 과정을 통해서 일단 소켓 연결이 맺어진 상태에서 메시지를 보내고자 할 때, 가장 먼저 보내지는 것이 데이터의 포맷이다. 프락시는 소스 노드인 프로세스 3의 포맷을 프로세스 0으로 전달하고, 자신의 상태를 *"await_format"*으로 바꾼 다음 프로세스 0으로부터 포맷 응답이 오기를 기다린다.
⑦ 프로세스 0은 프로세스 3의 포맷 메시지를 기다리고

있는 상태에서 포맷 메시지를 받으면, 자신의 포맷을 보내고 데이터에 대한 헤더 메시지를 기다리는 상태가 된다. 프락시는 프로세스 0으로부터 포맷 응답 메시지를 받으면 프로세스 3으로 이 메시지를 보내고, 자신의 상태를 *"await_header"*로 바꾼다.

⑧ 보낸 포맷에 대한 응답을 받으면, 소스 노드인 프로세스 3은 헤더 메시지를 Proxy에 보내고, Proxy는 다시 금 목적지 노드인 프로세스 0으로 헤더 메시지를 보낸다. 각각 Proxy와 프로세스 0의 상태는 데이터를 기다리는 상태인 *"await_data"* 상태가 된다.

⑨ 프로세스 3은 헤더 메시지를 보낸 후 바로 실제 데이터를 보낸다. 프락시와 목적지 노드인 프로세스 0은 데이터 메시지를 받으면 자신의 상태를 *"await_header"* 상태로 다시 바꾸고, 다음 데이터를 기다린다.

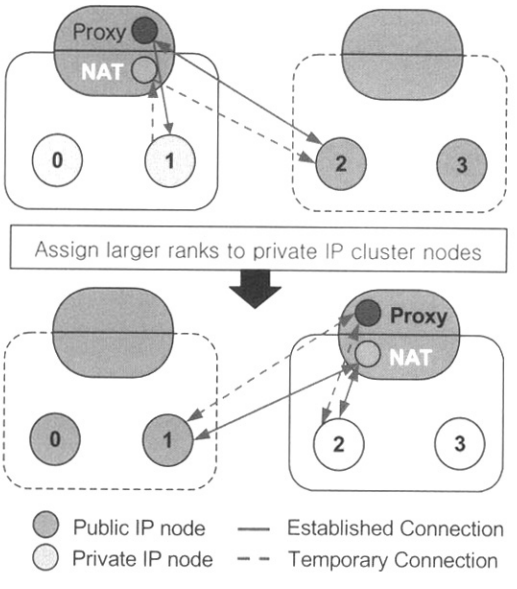
3.5 통신 중계를 위한 프락시

MPI 프로세스의 특징은 집합 연산을 통해 다른 프로세스들과 거의 동시에 연속적으로 자료를 교환한다는 것이다. 이에 따라 클러스터 내부의 노드 수가 많아지면 프락시는 전체 MPI 프로세스 작업의 진행을 저해하는 병목이 될 수밖에 없다. 그러므로 프락시를 설계함에 있어서 동시적인 클라이언트의 요청을 어떻게 처리할 것인가의 문제를 고려하여 클러스터 내부의 모든 노드로 들어오고 나가는 연속적인 요청들을 효과적으로 빠르게 처리할 수 있도록 해야 한다.

MPICH-GP의 프락시는 *select*를 사용하여 여러 동시적인 클라이언트의 요청을 다중으로 처리하는 하나의 프로세스로 구성된다. 이것은 글로버스가 제공하는 콜백 스페이스(Callback Space) 메커니즘을 이용한다. 콜백 메커니즘은 해당 콜백 스페이스를 정의하고, 입출력과 관련된 정보를 전부 가지고 있는 IO 핸들러를 콜백 스페이스에 등록하면, 콜백 스페이스에서 등록된 IO 핸들러를 감시하여 이벤트를 처리하는 방식으로 동작한다. 특정 소켓에서 이벤트가 발생하면 요청이 있는 파일 디스크립터와 입출력 핸들러를 인수로 넘겨주면서 등록된 콜백 함수를 호출하게 된다.

3.6 Rank 관리를 통한 성능향상 기법

(그림 10)의 프락시를 통한 MPI_Send/Recv 메시지의 중계 과정을 보면 통신을 위한 연결을 맺는 역할을 담당하는 *prime_the_line()*이라는 함수가 있다. 이 함수내의 두 노드 간의 연결 설정과정을 살펴보면, *global rank*를 비교하는 루틴이 포함되어 있다. 기존 MPICH-G2에서는 목적지 노드의 *global rank*가 소스 노드의 *global rank*보다 더 크면, 목적지 노드에게 다시 소스 노드로 연결을 맺도록 하고 자신은 연결을 끊어버린다. 공인 IP 클러스터로 구성되는 MPICH-G2의 경우 이러한 방식은 성능에 거의 영향을 미치지 않는다. 그러나 프락시를 사용하며 사설 IP 클러스터와 공인 IP 클러스터를 모두 지원하는 MPICH-GP의 경우 *global rank*에 따른 연결 설정 메커니즘은 경우에 따라 성능에 큰 영향을 미칠 수 있다. 사설 IP 클러스터만으로 구성된 네트워크에서 MPI 프로그램을 수행한다면 서로 다른 클러스터에 있는 노드 간에 통신을 위해서는 최소한 한번은 프락시를 거쳐야 한다. 그러나 사설 IP 클러스터와 공인 IP 클러스터가 혼합된 환경에서는 (그림 7)의 GP_GUID를 이용한 연결 설정 알고리즘에서 나타나듯이 경우에 따라서 프락시를 거치기도 하고 거치지 않기도 하는데, 이것은 목적지 노드의 *global rank*에 따라 달라진다.



(그림 11) Global Rank에 따른 연결 경로 변화

(그림 11)에서 보는 것처럼 기존 MPICH-G2에서의 연결 설정 메커니즘을 그대로 사용하게 되면, *global rank*가 1인 사설 IP 주소노드가 공인 IP 주소노드인 *global rank* 2와 통신하려고 하는 경우, 목적지가 더 높은 *global rank*를 가지고 있기 때문에 연결을 끊고 더 큰 *global rank*를 가진 노드로 하여금 다시 연결을 맺게 한다. 결과적으로 연결을 맺게 되는 경로를 살펴보면, 사설 IP 주소를 가지는 노드 1과 연결을 맺으려면 프락시를 통해야 하고 노드 1과 노드 2의 모든 메시지는 프락시를 통해 중계되어야 한다. 이것은 4장의 성능평가에서 살펴볼 수 있듯이 최초의 연결이 커널이 제공하는 NAT 서비스를 통해 중계되는 것과 비교하여 프록시를 거치는 오버헤드에 따른 성능 차이가 크다.

(그림 11)의 아래쪽 그림은 이 문제에 대한 해결책을 제시하고 있다. 사설 IP 주소노드에 더 큰 *global rank*를 가지도록 조정하게 되면 그림에서 보는 것과 같이, NAT 서비스를 통해서 공인 IP 주소를 가지고 있는 *global rank* 1인 노드로 직접 연결을 맺을 수 있다.

다음 장의 성능 비교를 통해서도 알 수 있지만, 일단 프락시를 한번이라도 통과하게 되면 그에 따른 오버헤드가 크기 때문에 가능하면 프락시를 통하지 않도록 하는 것이 좋다. 즉, 사설 IP 클러스터와 공인 IP 클러스터가 혼재하는 환경에서는 *global rank*를 할당하는 시점에서 사설 IP 클러스터 노드들에 대해서 공인 IP 클러스터 노드보다 높은 랭

〈표 2〉 MPICH-GP에서 나타나는 연결의 형태

MPI 구현	클러스터	연결의 종류(개수)
MPICH-GP(MPICH-G2)	공인 IP 주소 클러스터	두 프로세스 간 직접 TCP 연결
MPICH-GP	사설 IP 주소 클러스터	계산노드 - 프론트의 NAT - 상대 프론트의 프락시 - 상대 계산노드
MPICH-GP (RANK 관리 기법 적용 안한 경우)	공인 IP 주소 클러스터 + 사설 IP 주소 클러스터	계산노드(공인 IP 주소) - 상대 프론트의 프락시 - 상대 계산노드(사설 IP 주소)
MPICH-GP (RANK 관리 기법 적용)	공인 IP 주소 클러스터 + 사설 IP 주소 클러스터	계산노드(사설 IP 주소) - 프론트의 NAT - 상대 계산노드(공인 IP 주소)

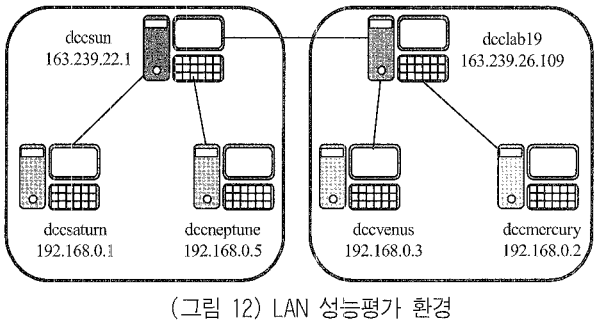
크를 가지도록 랭크를 할당함으로써 통신 중계에 따른 오버헤드를 줄일 수 있다.

4. 성능평가

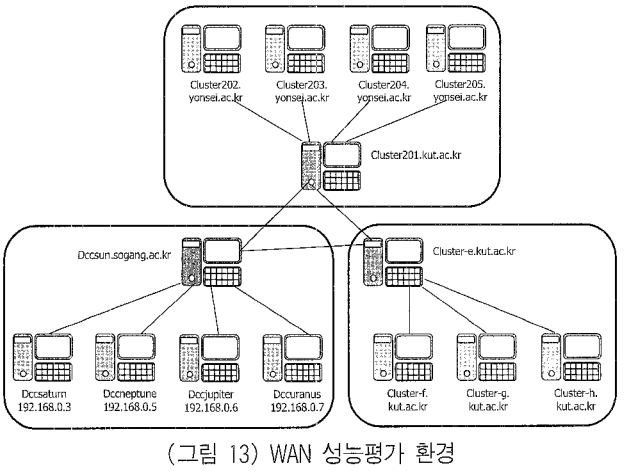
본 장에서는 MPI의 통신 프리미티브에 대하여 MPICH-G2와 MPICH-GP의 성능을 비교 측정하고, 이러한 프리미티브들이 복합적으로 작용하는 응용 프로그램들의 성능을 비교하면서 MPICH-GP의 성능을 측정해 보고자 한다. MPICH-GP는 프록시를 통해서 중계되는 경우와 RANK를 조절하여 NAT를 통해서 통신을 중계하는 형태로 나누어서 각각의 성능을 측정한다.

4.1 성능평가 환경

MPICH-GP는 기존의 MPICH-G2와 마찬가지로 다중 클러스터로 구성되는 그리드 환경에서의 실행을 목표로 하고 있다. LAN 환경에서의 성능평가를 통해서 네트워크의 상황



(그림 12) LAN 성능평가 환경



(그림 13) WAN 성능평가 환경

에 따른 영향이 없는 상태에서 순수하게 프락시를 통과하는 비용을 측정할 수 있으며, WAN 환경으로 확장하여 실제의 복합적인 환경에서 성능 편차가 어떻게 나타나는지 비교 분석하였다. (그림 12)와 (그림 13)은 성능평가를 위해 사용된 LAN과 WAN 환경의 구성도이다.

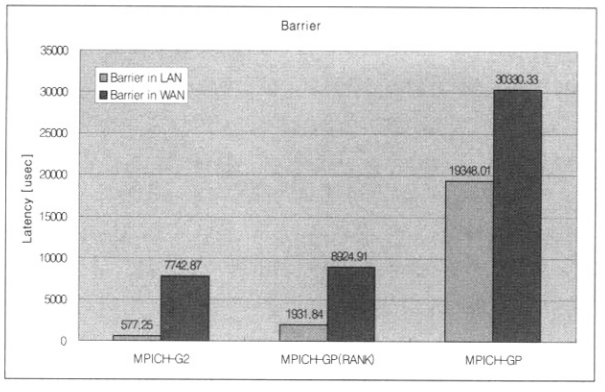
4.2 MPI 통신 프리미티브 성능 평가

Pallas MPI Benchmarks(PMB)[15]는 MPI 통신 프리미티브에 대한 성능 평가에 주로 사용된다. 본 실험에서는 MPI 통신 프리미티브 중 대표적 집합 연산(Collective Opearation)인 *MPI_Barrier*와 *MPI_Alltoall*에 대한 성능을 측정하였다.

MPICH-GP에서 나타나는 연결의 형태는 <표 2>와 같다. ①공인 IP 주소를 가지는 두 프로세스 간에는 직접 TCP 연결을 맺으며, ②사설 IP 주소를 가지는 두 프로세스는 NAT를 거쳐서 상대편 클러스터의 프론트 노드에 떠있는 프락시에 연결하고, 프락시가 실제 상대 프로세스로 연결을 맺는 총 3개의 통신 링크로 구성된다. ③사설 IP 주소 노드와 공인 IP 주소 노드 간의 통신인 경우 3.6장에서 살펴본 RANK를 조절 기법을 적용할 수 있는데, 이 경우에는 사설 IP 주소 노드와 프론트 노드의 NAT, 프론트 노드의 NAT와 상대 프로세스를 잇는 2개의 링크로 이루어진다. 본 실험에서는 ①의 연결을 갖는 MPICH-G2, ②에 해당하는 MPICH-GP, ③의 RANK 관리기법을 적용한 MPICH-GP(RANK)로 구분하여 이들의 성능을 비교분석 한다.

4.2.1 Barrier 테스트

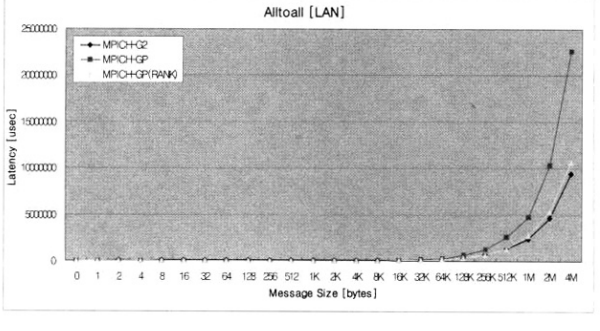
Barrier는 *MPI_Barrier*에 대한 벤치마크로서, 모든 프로세스들이 같이 다음 실행으로 넘어가고자 할 때 사용한다. (그림 14)는 LAN과 WAN에서 측정한 Barrier 벤치마크 결과이다. LAN에서 측정한 MPICH-G2의 경우는 어떠한 중계 장치도 거치지 않으므로 가장 빠르게 Barrier를 수행하며, MPICH-GP(RANK)는 모든 통신 메시지는 NAT를 거쳐야 하기 때문에 MPICH-G2보다 좀 더 오랜 시간이 걸린다. MPICH-GP의 경우는 프락시를 통해서 중계되기 때문에 LAN이라 할지라도 프락시 통과에 따른 지연시간은 기본적으로 동일하다. 이러한 이유로 LAN에서 측정한 MPICH-GP의 Barrier에 걸리는 시간은 다른 것들보다 크게 나타나지만 WAN에서 측정한 그래프에서는 지연시간의 증가가 크지 않은 것이다.



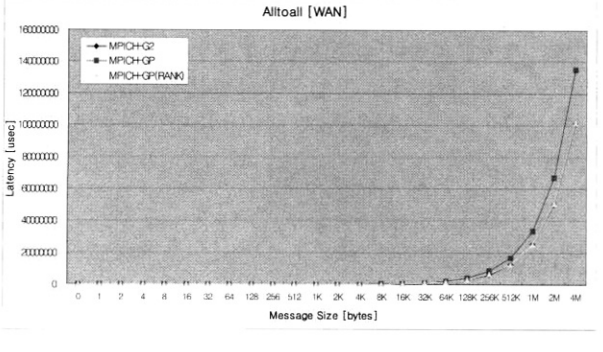
(그림 14) LAN & WAN에서의 Barrier 테스트

4.2.2 Alltoall 테스트

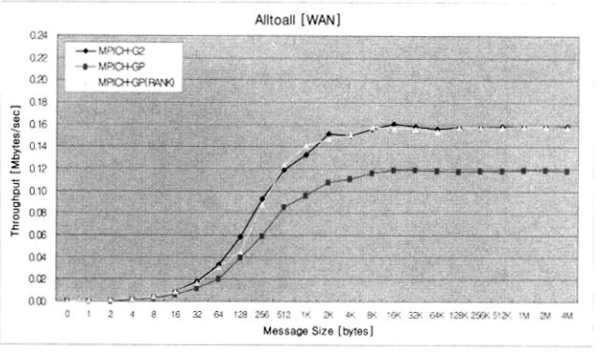
Alltoall 테스트는 *MPI_Alltoall*을 벤치마크 한다. 모든 프로세스는 각각의 프로세스를 위해 (각 프로세스에 대한 메시지 크기 * 프로세스의 수) 만큼을 입력하고 (각 프로세스로부터의 메시지 크기 * 프로세스의 수) 만큼을 받게 된다. 프락시는 클러스터의 프론트 노드에서 클러스터 내부의 모든 노드들로의 통신을 중계한다. 그렇기 때문에 클러스터 내부에 존재하는 노드들의 수가 많아지게 되면 그 모든 노드들의 연결을 처리해야 하는 프락시에 과도한 부하가 몰릴 수 있다. 특히 Alltoall 벤치마크는 프로세스가 N개의 존재한다면 전체 2^{N-1}개의 연결을 맺게 된다. 그렇기 때문에 다른 벤치마크들과 비교해서 상대적으로 프락시에 많은 부하를 주게 된다.



(그림 15) LAN에서의 Alltoall 테스트



(그림 16) WAN에서의 Alltoall 테스트



(그림 17) WAN에서의 Alltoall 테스트

(그림 15)의 그래프는 LAN상의 두 클러스터에서 각각 두개의 노드씩 전체 네 개의 노드에서 Alltoall 벤치마크를 수행한 결과이다. 본 그래프에서는 메시지의 크기가 2배씩 증가함에 따라 지연시간은 지수적인 증가를 보이는데, 이것은 메시지 크기가 X일 때 한 프로세스가 다른 프로세스로 보내는 실제 메시지는 4X (메시지 사이즈 X * 프로세스의 개수)이기 때문이다. 그래프에서 보면 NAT를 통해 메시지를 중계하는 RANK 관리기법을 적용한 MPICH-GP의 경우는 MPICH-G2와 거의 비슷한 지연시간을 보이고 있지만, 프락시를 통해 메시지를 중계하는 MPICH-GP의 경우, 프락시로 과도한 메시지 중계에 따른 부하가 몰리기 때문에 이와 같이 지연시간이 가파르게 증가하는 그래프로 나타난다.

(그림 16)의 그래프는 WAN상의 두 클러스터의 네 개의 노드에서 Alltoall 벤치마크를 수행한 결과이다. (그림 15)와 비교해서 전체 지연시간이 크게 증가하였으며, MPICH-G2와 MPICH-GP 사이의 간격이 확연히 줄었음을 알 수 있다.

(그림 17)은 (그림 16)의 지연시간 그래프를 처리량 그래프로 변환한 것으로, WAN 상에서는 MPICH-GP(RANK)의 성능이 MPICH-G2와 동일한 성능을 나타내고 있다. MPICH-GP의 경우는 MPICH-G2의 75% 정도의 성능을 보인다.

4.3 응용 프로그램 성능 평가

본 실험에서는 MPI의 여러 통신 프리미티브들이 조합되어 사용되고 있는 응용 프로그램에 대한 MPICH-GP의 성능을 분석한다. 이를 위해서 NAS Parallel Benchmarks(NPB)[16] 3.1 버전을 사용하였다. NPB 벤치마크에는 LU, CG, IS, FT, MG, EP, SP의 7개의 벤치마크 프로그램이 포함되어 있으며 이들 각각은 다양한 통신 패턴을 표현하고 있다.

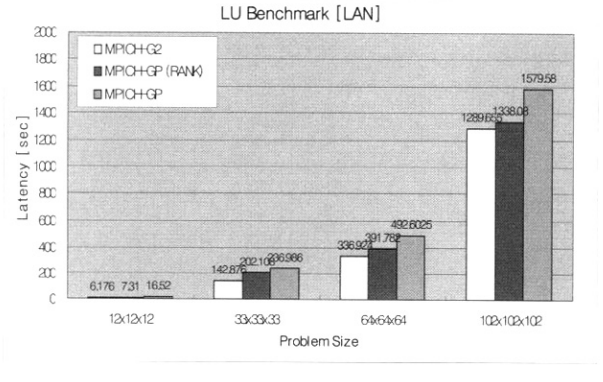
응용 프로그램이 이웃 노드들과의 통신이 많고 외부 클러스터 노드와의 통신이 적은 경우에, 외부 클러스터 노드와의 통신을 중계하는 프락시의 부담을 측정하기 위해 LU 벤치마크가 좋은 성능을 낼 수 있는지에 관한 실험을 진행하였다. 네트워크의 영향을 고려하지 않고 문제의 크기에 따른 응용 프로그램의 성능을 비교하기 위해서 LAN 상에서 CG, LU에 대한 벤치마크를 수행하였다.

〈표 3〉 NPB 벤치마크 특성

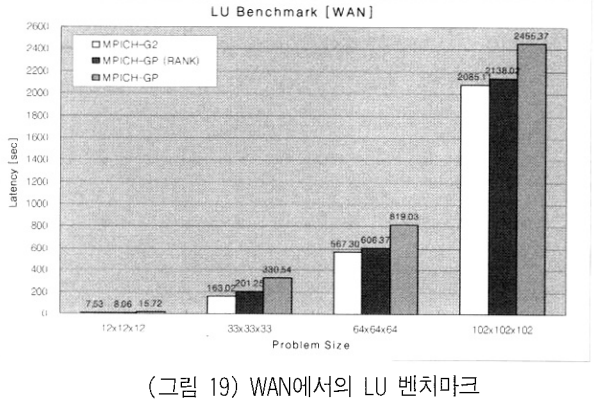
벤치마크	문 제 크 기	벤치마크의 특성
LU	클래스 S : 12x12x12	작은 크기의 메시지를 주고받으며 가까운 노드들과의 통신에 의존한다.
	클래스 W : 33x33x33	
	클래스 A : 64x64x64	
	클래스 B : 102x102x102	
CG	클래스 S : 1400	노드들 간에 통신량이 많다.
	클래스 W : 7000	
	클래스 A : 14000	
	클래스 B : 75000	

4.3.1 LU 벤치마크

(그림 18)은 LAN 상의 *dccsun* 클러스터와 *dcclab19* 클러스터의 4대의 노드에서 LU 벤치마크의 결과이다. 클래스 S의 경우 MPICH-GP와 MPICH-G2의 지연시간은 2.67로 나타나지만 문제의 크기가 커짐에 따라 그 비율이 점차 줄어서 클래스 B에 해당하는 102x102x102x102에서는 1.22로 MPICH-GP의 성능이 MPICH-G2의 80% 정도로 나타난다. 이것은 MPICH-GP의 성능이 MPI 통신 프리미티브에서 측정한 결과보다 실제 응용 프로그램에서 좋은 성능을 낼 수 있음을 보여주는 것이다. MPICH-GP(RANK)는 MPICH-G2와 지연시간이 거의 동일한 비율로 증가하는 형태를 보이며 LU 벤치마크의 경우에는 MPICH-G2의 95% 이상의 성능을 보인다.



(그림 18) LAN에서의 LU 벤치마크

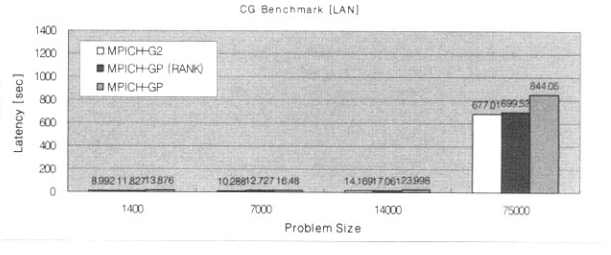


(그림 19) WAN에서의 LU 벤치마크

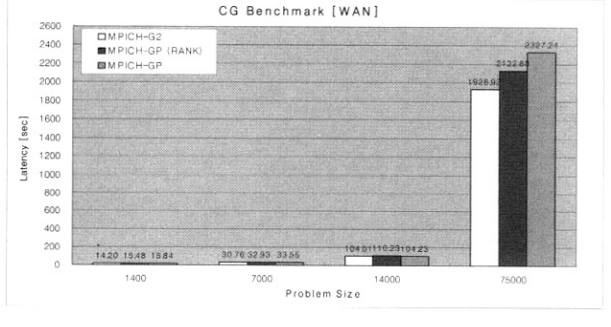
(그림 19)는 WAN 상의 두 개의 클러스터의 4대의 노드에서 측정한 LU 벤치마크의 결과이다. (그림 18)와 비교하여 WAN 환경으로 옮겨감에 따라 전체적으로 지연시간이 늘어난다. 각 클래스별로 보면 LAN에서의 MPICH-GP/MPICH-G2의 비율이 줄어들었음을 확인할 수 있다. LAN에서는 클래스 S의 경우 2.66을 보였으나 WAN에서는 2.09를 보인다. 이것은 프락시의 통신 중계에 따른 지연이 WAN의 네트워크 지연 정도에 따라서 성능 저하의 영향력을 둔화시킴을 보여주는 예라 할 수 있다.

4.3.2 CG 벤치마크

(그림 20)은 LAN 상의 두 클러스터의 4대의 노드에서 CG 벤치마크의 결과이고, (그림 21)는 WAN상의 두 클러스터, dccsun과 cluster-e의 4개 노드에서 CG 벤치마크 그래프이다. LAN에서 측정한 그래프와 WAN에서의 그래프를 비교하여 보면 비교하여 보면 WAN의 네트워크 지연시간이 대략 3배 정도 크다.



(그림 20) LAN에서의 CG 벤치마크



(그림 21) WAN에서의 CG 벤치마크

작은 메시지의 경우에서 나타나는 지연시간 비의 큰 차이는 네트워크의 지연이 없는 LAN 상에서 프락시를 통과하는데 걸리는 시간이 전체 지연시간 중에서 큰 비중을 차지하기 때문이라 볼 수 있다. MPICH-GP(RANK)는 NAT를 거쳐서 메시지를 중계하는 RANK 관리기법을 적용한 MPICH-GP와 MPICH-G2의 지연시간 비율로서 이 경우에는 커널을 통한 메시지 중계이므로 지연시간 차이가 더 작게 나타남을 볼 수 있다.

WAN에서 테스트한 CG의 결과는 LAN에서의 평가와 비교하여 네트워크 속도 지연은 3배 이상 나타나고 있다. 이에 따라서 MPICH-GP에서의 성능이 LAN에서 보다 좋은

결과를 나타낸다. 프락시를 통해 중계되는 MPICH-GP의 경우 연결의 수가 많아질수록, 그리고 메시지 사이즈가 커질수록 프락시에서 모든 연결을 중계해야 하므로 부하는 높아진다. NAT를 통한 중계의 경우에도 커널 영역에서 구현되었다 하더라도 프론트 노드로 연결이 집중되기 때문에 지연시간은 늘어나게 된다.

5. 결론 및 향후 과제

그리드 네트워크에서 MPI를 사용하여 지리적으로 산재한 컴퓨팅 자원을 활용하여 복잡한 문제를 해결하기 위해 구현된 MPICH-G2는 사설 IP 클러스터를 지원하지 못한다는 단점을 가지고 있다. 본 논문에서는 MPICH-G2가 가지는 이러한 문제점을 해결하는 방법으로서 사설 IP 클러스터를 지원하는 MPICH-GP를 설계하여 구현하였다.

MPICH-GP는 NAT와 프락시를 병용하여 사설 IP 클러스터의 문제를 해결한다. 사설 IP 클러스터의 프론트 노드에 프락시를 두고, 이를 통해 내부 계산 노드로의 통신 링크를 중계한다. 계산에 참여하는 사설 IP 주소노드와 공인 IP 주소노드에 GP_GUID를 할당하여 이를 통해 사설 IP 주소노드와 공인 IP 주소노드를 구별하고 그에 따라 적정한 경로를 선택하여 연결을 맺고 통신을 하는 방식을 통해서 사설 IP 클러스터와 공인 IP 클러스터가 혼재된 네트워크에서도 MPI 작업을 수행할 수 있다.

본 논문의 성능 평가 부분에서는 현재 가장 널리 사용되고 있는 MPICH-G2와 MPICH-GP, 그리고 RANK 관리기법을 적용한 MPICH-GP의 성능을 PMB[15]와 NPB[16]를 이용하여 비교 측정하였다. LAN에서의 성능 측정 결과 조금씩 차이는 있지만 RANK 관리기법을 적용한 MPICH-GP(RANK)의 경우는 MPICH-G2 성능의 90%, 적용하지 않은 MPICH-GP의 경우는 MPICH-G2의 50% 정도의 성능을 보였다. 동일한 통신 프리미티브와 응용 프로그램에 대하여 WAN에서 실험한 결과, 네트워크에서의 전파지연과 혼잡도가 LAN에서보다 훨씬 크기 때문에 통신 중계 장치인 프락시와 NAT에서 걸리는 지연시간은 어느 정도 가려지게 된다. 실제로 MPICH-GP(RANK)의 경우는 MPICH-G2 성능의 95% 이상의 성능을 보였으며, MPICH-GP의 경우는 평균적으로 MPICH-G2의 80% 가량의 성능을 나타낸다.

MPICH-GP는 그리드 환경에서 사설 IP 클러스터 문제를 해결한다는 점에서 기여를 하지만, 실제적으로 폭넓게 이용되기 위해서는 성능상의 개선이 필요하다. 한 가지 방안으로 일정 크기 이상의 메시지에 대해서는 압축을 해서 보내는 방식을 생각해볼 수 있다. 실제 이를 적용하기 위해 시도하였으나, 라이브러리에 메시지에 적합한 압축 알고리즘의 선택과 적용이 어려웠기 때문에 실험에서는 제외시켰다. 또한 다수의 계산노드로 구성되는 클러스터의 경우, 프론트 노드에 존재하는 하나의 프락시로 전체 노드들로의 연결을 관리하고 메시지를 처리한다는 것은 무리가 따른다. MPICH-GP의 성능을 개선하기 위해서는 프락시로 몰리는 부하를 분산시키기 위한 장치가 필요하며 이에 대한 연구가 필요하다. 성능상의 문제를 해결하는 것과 더불어 실행상의 문제를 해결한다. 현재 많은 시스템들이 방화벽으로 묶여 있기 때문에 실행 상에 문제가 있다. MPI를 위한 프락시에 방화벽을 넘어서 통신할 수 있도록 하는 것이 향후 과제이다. NAT나 방화벽과 환경에서의 통신을 가능하게 해주는 Hole Punching 기법[17]은 이러한 노력중의 하나로 볼 수 있을 것이다.

참 고 문 헌

[1] I. Foster, C. Kesselman and S. Tuecke. "The Anatomy of the grid : Enabling scalable virtual organizations". International Journals of Supercomputing Applications, 15(3), 2001

[2] I. Foster and C. Kesselman. "The Grid : A Blueprint for a New Computing Infrastructure". Morgan Kaufmann, 1998

[3] W. Gropp, E. Lusk, and A. Skjellum. "Using MPI: Portable Parallel Programming with the Message Passing Interface". MIT Press, 1995.

[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. "A high-performance, protable implementation of the MPI message passing interface standard." Parallel Computing, Volume 22, pp.789-828, 1996.

[5] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. "A wide-area implementation of the Message Passing Interface." Parallel Computing, pp. 1735-1749, 1998.

[6] N. Karonis, B. Toonen, I Foster, "MPICH-G2: a Grid-enabled implementation of the Message Passing Interface", Journal of Parallel and Distributed Computing, Volume 63, pp.551-563, 1998.

[7] P. Srisuresh and K. Egevang. "Traditional IP Network Address Translator(Traditional NAT)", RFC 3022. International Engineering Task Force, Jan, 2001.

[8] M. Műller, M. Hess, E. Gabriel, "Grid enabled MPI solutions for Clusters", In 3rd International Symposium on Cluster Computing and the Grid, pp.18-25, 2003.

[9] Globus I/O API, "Globus-IO reference", http://www.globus.org/v1.1/io/globus_io.html

[10] E. Gabriel, M. Resch, T. Beisel and R. Keller, "Distributed computing in a heterogeneous computing environment", In Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Nodes in computer Science. Springer, 1497, pp.180-197, 1998.

[11] Y. Tanaka, M. Sato, M. Hirano, H. Nakada, and S. Sekiguchi. "Performance evaluation of a firewall-complaint globus-based wide-area cluster system." In Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing, pp. 121-128. IEEE Computing Society, 2000.

[12] I. foster and C. Kesselman, and S. Tuecke. "The Nexus approach to intergrating multithreading and communication. Journal of Parallel and Distributed Computing", pp.70-82, 1996.
[13] Globus DUROC, http://www.globus.org/duroc/frames.html
[14] S. Choi, K. Park, S. Han, S. Park, "An NAT-Based Communication Relay Scheme for Private-IP-enabled MPI over Grid Environments", International Conference on Computational Science 2004 (ICCS 2004), pp 499-502, 2004
[15] Pallas MPI Benchmarks, http://www.pallas.com/e/products/pmb/
[16] NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB
[17] Ford Bryan, Srisuresh Pyda, Kegel Dan. "Peer-to-Peer Communication Across Network Address Translators", USENIX Annual Technical Conference, 2005.

박 금 례

e-mail : kumrye.park@samsung.com
2005년 서강대학교 컴퓨터학과 (공학석사)
2005년~현재 삼성전자 정보통신총괄 무선사업부 WiBro S/W Lab
관심분야 : WiBro, Mobile WiMAX, 4G

윤 현 준

e-mail : hjyun@dcclab.sogang.ac.kr
2006년 서강대학교 컴퓨터학과(공학사)
2006년~현재 서강대학교 컴퓨터학과 석사과정
관심분야 : High Performance Cluster Computing and System

박 성 용

e-mail : parksy@sogang.ac.kr
1987년 서강대학교 컴퓨터학과(공학사)
1994년 미국 Syracuse University 대학원 (공학석사)
1998년 미국 Syracuse University (공학박사)
1998~1999 미국 Bell Communication Research 연구원
1999~현재 서강대학교 컴퓨터학과 조교수 / 부교수
관심분야 : Autonomic Computing, Peer to Peer Computing, High Performance Cluster Computing and System

권 오 영

e-mail : oykwon@kut.ac.kr
1997년 연세대학교 컴퓨터과학과 (공학박사)
1997년~2000년 ETRI 선임연구원
2000년~현재 한국기술교육대학교 부교수
관심분야 : High-performance computing, Embedded middleware for home networking and sensor networking

권 오 경

e-mail : okkwon@kisti.re.kr
2000년 고려대학교 컴퓨터학과(공학사)
2002년 KAIST 전산학과(공학석사)
2001년 12월~현재 KISTI 슈퍼컴퓨팅 센터 연구원
관심분야 : Grid Computing, High-performance computing