韓國 컴퓨터情報學會 論文誌
第13卷 第2號, 2008. 3.

2008-13-2-3-7

PC 클러스터를 위한 TCP/IP 기반 하이퍼큐브 네트워크 구현

이 형 봉*, 홍 준 표**, 김 영 태*

Implementations of Hypercube Networks based on TCP/IP for PC Clusters

Hyung-Bong Lee*, Joon-Pyo Hong**, Youngtae Kim*

요 약

일반적으로 병렬처리가 필요한 경우 병렬처리 전용으로 제조된 시스템을 사용하지만, 가까운 주위에서 쉽게 얻을 수 있는 PC들을 클러스터로 구축하여 병렬처리에 활용할 수도 있다. PC들을 클러스터로 구축하기 위한 가장 쉬운 방법은 PC들을 스위치 허브 중심의 스타 네트워크로 연결하는 방법이지만, 이 논문에서는 병렬처리 연구 및 활용에 더 적합하도록 8 개의 PC들이 직접 연결된 클러스터 구축을 위한 TCP/IP 기반 하이퍼큐브 네트워크의 효율적인 구현 방안을 모색하고, 그 기능 및 효율성을 ping, netperf, MPICH 등의 도구를 이용하여 검증하였다. 구현 방안으로 링크 위주의 IP 설정 방법과 노드 위주의IP 설정 방법을 제안하고 두 방법을 비교·분석하였는데, 그 결과 두 방법간에 시간적 성능 차이는 없지만 라우팅 테이블의 단순화 측면에서 노드 위주의 IP 설정 방법이 우수함을 볼 수 있었다. 기능적 측면을 검증하기 위하여 응용프로그램의 병렬처리 결과를 스타 네트워크 기반 클러스터에서의 결과와 비교하였는데, 두 방법 모두 완벽한 병렬처리 환경을 지원하는 것으로 나타났다.

Abstract

In general, we use a parallel processing computer manufactured specially for the purpose of parallel processing to do high performance computings. But we can deploy and use a PC cluster composed of several common PCs instead of the very expensive parallel processing computer. A common way to get a PC cluster is to adopt the star topology network connected by a switch hub. But in this paper, we grope efficient implementations of hypercube networks based on TCP/IP to connect 8 PCs directly for more useful parallel processing environment, and make evaluations on functionality and efficiency of them using ping, netperf, MPICH. The two proposed methods of implementation are IP configuration based on link and IP configuration based on node. The results of comparison between them show that there is not obvious difference in performance but the latter is more efficient in simplicity of routing table. For verification of functionality, we compare the parallel processing results of an application in them with the same in a star network based PC cluster. These results also show that the proposed hypercube networks support a perfect parallel processing environment respectively.

▸ Keyword : 병렬처리(Parallel Processing), 클러스터(Cluster), 하이퍼큐브 네트워크(Hypercube Network)

• 제1저자 : 이형봉 교신저자 : 김영태
• 접수일 : 2008. 2. 5, 심사일 : 2008. 2. 12, 심사완료일 : 2008. 3. 8.
* 강릉대학교 컴퓨터공학과 교수 **강릉대학교 컴퓨터공학과 석사과정, (재)사회서비스관리센터

Ⅰ. 서 론

PC 에 사용되는 CPU의 계산 속도가 급속하게 진전함에 따라 PC들을 네트워크로 연결하여 병렬처리가 가능하도록 한 PC 클러스터가 고가의 상업용 슈퍼 컴퓨터를 대체할 수 있을 정도가 되었다[1]. 실제로 2007년 11월 현재 세계적인 상위 500대 슈퍼 컴퓨터 중 70개 이상이 인텔 펜티엄 제온 (Intel Pentium Xeon) 프로세서로 구성되어 있다[2].

다양한 산업 및 연구 분야에서 더욱 빠르고 안정된 고성능 컴퓨터에 대한 수요가 증대되고 있으나 병렬처리 전용으로 제작된 수퍼 컴퓨터는 고가이기 때문에 도입에 어려움이 있다. 따라서, 활용 분야에 따라서는 그러한 고가의 수퍼 컴퓨터보다는 가까운 주변에서 쉽게 얻을 수 있는 PC들을 클러스터로 구축하여 활용할 경우 수퍼 컴퓨터에 못지않은 큰 효과를 얻을 수 있다.

PC 클러스터의 성능은 크게 CPU의 계산 속도와 PC사이의 네트워크 속도에 의존하게 되는데, 최근 CPU 발전 속도에 비해 네트워크 발전 속도가 상대적으로 저조한 현상을 보이고 있다. 따라서, 효율적인 네트워크의 선택은 PC 클러스터의 구축에 있어서 가장 중요한 요인으로 인식되고 있다

PC 클러스터를 구축하는 가장 간편한 방법은 중앙에 위치한 스위치 허브를 중심으로 PC들을 스타(star)형으로 연결하는 네트워크를 사용하는 것인데, 이 때 스위치 장비로 주위에서 흔히 볼 수 있는 허브를 사용할 수 있다.

이 때 스위치 허브 장비로 GigEthernet가 대표적이고, Myrinet, Quadrics 등의 스위치가 사용된다[2,3,4].

이러한 네트워크 장비들은 쉽게 구축할 수 있고 비교적 효율적인 성능을 보이지만, 장비 자체에서 최소한의 제어 과정을 필요로 하기 때문에 PC들을 직접 연결한 경우보다는 성능이 높지 않을 수 있다. 또한 네트워크 장비에 대한 의존도가 높아져서 해당 장비에 문제가 발생할 경우 클러스터 전체의 신뢰성을 잃게 된다.

그 반면에, PC들을 직접 연결하는 하이퍼큐브(hypercube) 네트워크를 사용하는 PC 클러스터는 별도의 장비를 사용하지 않기 때문에 비용이 저렴하고, 8개 정도의 비교적 적은 수의 PC를 사용할 경우에는 PC간의 간접 통신 부분으로 인한 부하보다 직접 통신 부분에 의해 얻어지는 이득이 더 커서 전체적인 성능이 우수할 수 있다. 또한 PC들 사이의 통신 경로가 하나 이상이므로 네트워크 일부에 오류가 발생하더라도 클러스터 전체가 기능을 잃는 파국 상황을 예방할 수 있다.

이 논문에서는 LAN 카드 등 연구 환경의 제한으로 절대적 성능 측면보다는 병렬처리 알고리즘 연구나 실습 등을 목적으로 주변에서 하이퍼큐브 네트워크기반 클러스터를 쉽고 저렴하게 얻기 위해 PC들을 직접 연결하는 TCP/IP 기반 하이퍼큐브 네트워크의 구현 방안들을 제시하고, 그 기능적 타당성을 스타 네트워크 기반 PC 클러스터와 비교분석함으로써 검증하고자 한다.

이를 위해, 2장에서 하이퍼큐브 네트워크 및 관련 연구를 조명해보고, 3 장에서 TCP/IP프로토콜을 사용한 하이퍼큐브 네트워크 구축을 위한 리눅스 환경 설정 방안들을 제안하며, 4 장에서 몇 가지 도구를 사용하여 설정 방안들 사이의 시간적 성능을 분석해 보고, 스타 네트워크 기반 PC 클러스터와의 비교를 통해 기능적 타당성을 검증하며, 마지막 5 장의 결론으로 이 논문을 맺는다.

Ⅱ. 관련연구

2.1 하이퍼큐브 네트워크의 구조

일반적으로 컴퓨터 통신은 크게 네트워크 장비를 경유하는 간접 연결 방식과 네트워크 장비를 필요치 않는 직접 연결 방식으로 구분될 수 있다. 네트워크 장비를 거치는 가장 단순한 연결 방식은 〈그림 1〉과 같이 한 개의 스위치 허브를 중심으로 모든 컴퓨터를 연결하는 스타 네트워크인데, 고성능 스위치 장비로 GigEthernet [2,3,4]이 잘 알려져 있으나 일반적인 스위치 허브를 사용해서 PC 클러스터를 구현할 때 손쉽게 이용될 수 있다. 이 방식은 통신 속도가 일정하고 시스템 구성이 용이하다는 장점이 있으나, 노드의 개수가 늘어날수록 전체적인 통신 성능이 급격하게 저하되고, 특정 네트워크 장비에 의존하게 된다는 단점이 있다[4]. 크로스바(crossbar) 네트워크는 컴퓨터를 논리적인 x-y 축으로 배치하여 교차 지점에 스위치 장비를 배치하는 방식으로 스타 네트워크 보다 네트워크 장비로 인한 부하를 줄일 수 있으나, Myrinet[3]과 같은 전문 스위치 장비를 사용해야 하는 단점이 있다.

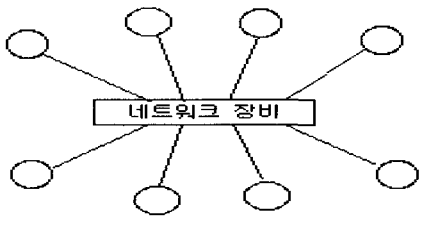

그림 1. 스타 네트워크 방식
Fig 1. Connection of Star Network

네트워크 장비를 통하지 않고 컴퓨터를 점대점(point-to-point) 형태로 직접 연결하는 방식은 네트워크 장비를 거쳐야 하는 부하가 제거되는 최선의 장점을 가지고 있다. 그러나, 노드가 많아질수록 연결 경로가 증가하여 확장성이 좋지 않다는 단점이 있다. 이 방식에는 완전연결(fully connected), 하이퍼큐브 등의 네트워크가 있다 [5,6].

완전연결 네트워크는 모든 노드가 직접 연결되어 성능은 뛰어나지만 nC2(n:노드 수)개의 많은 연결을 필요로 하기 때문에 네트워크 구성 비용이 크다. <그림 2>의 하이퍼큐브 네트워크는최소 연결 구조로 네트워크를 구성할 수 있다. 즉, 2n-1개의 연결을 필요로 하기 때문에 8대 정도의 비교적 적은 수의 컴퓨터를 연결할 때는 매우 효율적인 것으로 알려져 있다. <그림 2>에는 8노드와 16 노드를 위한 각각의 하이퍼큐브 네트워크를 보였다. 이 그림에서 보는 바와 같이 하이퍼큐브 네트워크에서 노드 id는 그레이 코드(gray code)를 활용하여 부여함으로써 두 노드 간의 거리를 쉽게 알 수 있도록 한다. 예를 들어, 16노드 하이퍼큐브 네트워크에서 노드 0000 과 노드 0010사이의 거리는 1, 노드 0000과 0101 사이의 거리는 2, 노드 0000과 노드 1011 사이의 거리는 3, 노드 0000과 노드 1111 사이의 거리는 4 이다. 이 때 거리가 1이라는 것은 직접 통신이 가능하고, 거리가 4라는 것은 3개의 노드를 거쳐 간접적으로 통신할 수 있음을 의미한다.

2.2 하이퍼큐브 네트워크 관련연구

하이퍼큐브 네트워크와 관련된 연구는 하이퍼큐브 네트워크 환경하에서의 효율적인 전송 기법이나 병렬처리를 위한 효율적인 알고리즘 제안에 관한 내용이 대부분이고, 이 논문과 같이 PC클러스터를 위한 TCP/IP 기반 하이퍼큐브 네트워크 구현 방안에 관한 내용은 매우 드물다. 본 논문과 관계가 깊은 [4]에서 스타 네트워크를 기반으로 하는 PC 클러스터를 구축하여 스위치 허브 성능이 클러스터 전체 성능에 미치는 영향을 분석하였는데, 하이퍼큐브 네트워크에는 적용하지 못했다. 그 밖에 [7]은 하이퍼큐브 네트워크에서 오류가 발생할 가능성이 있는 구성 요소들을 전제로 다중 전송함으로써 결국은 하이퍼큐브 네트워크의 전체적인 통신 성능(throughput)을 향상시키는 방안을 제안하였는데, 고차원 하이퍼큐브 네트워크에서는 그 효과가 미미하다. [8]은 하이퍼큐브 및 스타 네트워크가 혼재하는 하이퍼-스타 네트워크에서 나타나는 위상적 성질을 규정하여 하이퍼-스타 그래프 HS(m, k)를 도출하고 이를 이용한 효율적인 방송 알고리즘을 제안하 였다.

III. PC 클러스터를 위한 하이퍼큐브 네트워크의 설계 및 구현

하이퍼큐브 네트워크 기반 PC 클러스터의 기능적 성능 분석을 위해 하이퍼큐브 네트워크 기반 PC 클러스터와 더불어 스타 네트워크 기반 PC 클러스터도 함께 구현하였다.

3.1 PC 시스템 환경

■ 기본 시스템

[표 1]에 하이퍼큐브 네트워크 및 스타 네트워크 기반 PC 클러스터에 사용된 PC들의 공통 기본 시스템 사양을 보였다.

표 1. 클러스터에 사용된 PC 시스템 사양
Table 1. Specifications of PC

구 성 항 목	사 양	비 고
모 델	직 접 조 립	8대 동일
C P U	Intel Pentium IV 3.0GHz	
메 모 리	512 MB	
운 영 체 제	Linux 2.6.9-1.667	

■ 네트워크 장치 및 통신 프로토콜

[표 2]에 하이퍼큐브 네트워크 및 스타 네트워크 기반 PC 클러스터 각각에 사용된 네트워크 장치와 통신 프로토콜의 사

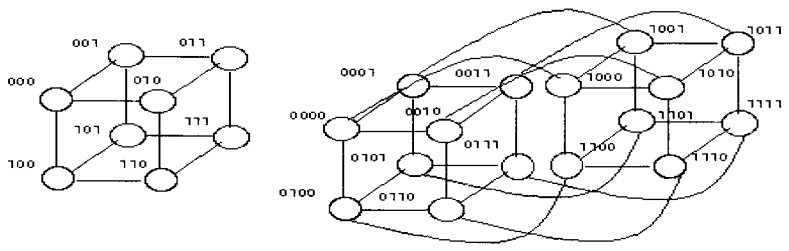

그림 2. 하이퍼큐브 네트워크 연결 방식
Fig 2. Connection of Hypercube Network

양을 보였다. 이 표에서 모두 동일한 LAN 카드를 장착하지 못한 이유는 Gigabit PCI LAN 카드를 더 이상 구하지 못했기 때문인데, 이 논문에서는 주로 하이퍼큐브 네트워크를 구현하는 기능적 측면에 관심이 있으므로, 기능 검증의 비교 대상으로 사용된 스타 네트워크에 다른 유형의 LAN 카드를 사용해도 무방하다.

표 2. 클러스터에 사용된 네트워크 장치 사양
Table 2. Specifications of Network devices

구 성 항 목	스 타	하 이 퍼 큐 브
LAN 카드	Intel Pro/1000 MT PWLA8390MT-LP	Gigabit PCI Adapter(Netgear) 10/100/1000 Mbps Gigabit Ethernet GA31
스위치(허브)	Gigabit Ethernet Switch Series JGS500	직 접 연 결
LAN 케이블	1000base-T(UTP)	1000base-T(UTP)
통신프로토콜	TCP/IP	TCP/IP

3.2 TCP/IP 기반 하이퍼큐브 네트워크 설계 · 구현

통신 프로토콜로TCP/IP를 사용하기 때문에 [표 1], [표 2]에 보인 PC 8대를 직접 연결한 3차원 하이퍼큐브 네트워크 기반 PC 클러스터를 구축하기 위해서는 TCP/IP 규약을 만족하면서도 통신 부하를 균일하게 분산시킬 수 있는 조심스러운 네트워크 설정이 필요하다.

3.2.1 PC 연결

■ 이더넷 케이블 연결

각 PC에 [표 2]에 보인 3 개의 LAN 카드를 장착하고 1000 base-T 크로스 케이블을 사용하여 각각에 인접한 3 대의 서로 다른 PC와 직접 연결한다. 각 PC마다 3 개의 연결이 필요하고 이들 각 연결은 양쪽 PC가 공유하므로 전체 하이퍼큐브 네트워크를 위해서는 총 12(=3x8/2)개의 연결이 필요하다.

■ 노드(PC) id

하이퍼큐브 네트워크 상에서 각 노드의 id는 〈그림 2〉에 보인 그레이 코드 값을 10진수로 변환한 값을 적용하여 node0~node7로 부여한다.

3.2.2 노드 인터페이스(IP 주소) 설정

어느 한 노드를 기준으로 3 개의 독립된 인터페이스(LAN 카드)가 존재하므로 해당 노드는 3 개의 독립된 IP 주소의 설정이 필요한데, 이 논문에서는 링크 위주의 설정 방법과 노드 위주의 설정 방법 등 두 가지 구현 방안을 제시한다.

3.2.2.1 링크 위주의 IP 주소 설정 방안

잘 알려진 바와 같이 TCP/IP 주소는 net id와 host id 등 두 부분으로 구성되는데, 이 때 net id는 분리된 물리적 전송매체 단위로 부여되는 유일한 id이고, host id는 특정 net id를 가진 전송매체(링크)를 공유하는 호스트들에게 부여되는 id로 해당 net id 내에서 유일하다[9]. 〈그림 3〉에서 하이퍼큐브 네트워크 연결을 위한 분리된 전송매체는 총 12개이고 각 매체는 2 개의 호스트가 공유하므로 12개의 net id와 각 net id 별 2 개의 host id가 필요하다. 예를 들어 node0과 node1 사이에 링크 하나가 존재하며, 양쪽에 두 인터페이스 a와 o가 존재한다. 이와 같은 전송매체 위주의 TCP/IP 주소체계에 바탕을 둔 주소부여 방안을 [표 3], 〈그림 4〉에 보였다(내부 IP 주소 도메인에 net mask[9]를 0xffffff00로 연장).

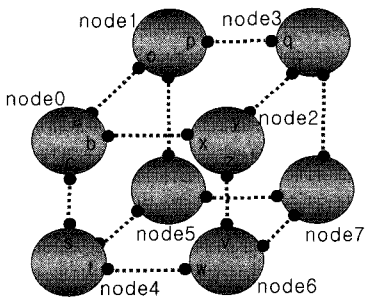

그림 3. 노드 인터페이스 IP 주소 설정
Fig 3. IP configuration of node interfaces

표 3. 노드 위주의 IP 주소 설정 방안
Table 3. IP configuration based on node

노드(host id)	인터페이스 net id [device]	노드(host id)
node0(101)	[eth1] 192.168.101 [eth1]	node1(110)
node0(102)	[eth2] 192.168.102 [eth1]	node2(120)
node0(104)	[eth3] 192.168.104 [eth1]	node4(140)
node1(113)	[eth2] 192.168.113 [eth1]	node3(131)
node1(115)	[eth3] 192.168.115 [eth1]	node5(151)
node2(123)	[eth2] 192.168.123 [eth2]	node3(132)
node2(126)	[eth3] 192.168.126 [eth1]	node6(162)
node3(137)	[eth3] 192.168.137 [eth1]	node7(173)
node4(145)	[eth2] 192.168.145 [eth2]	node5(154)
node4(146)	[eth3] 192.168.146 [eth2]	node6(164)
node5(157)	[eth3] 192.168.157 [eth2]	node7(175)
node6(167)	[eth3] 192.168.167 [eth3]	node7(176)

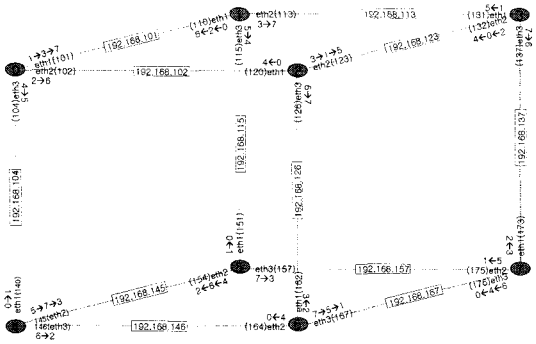

그림 4. [표 3]에 의한 하이퍼큐브 네트워크
Fig 4. Hypercube network based on Table 3

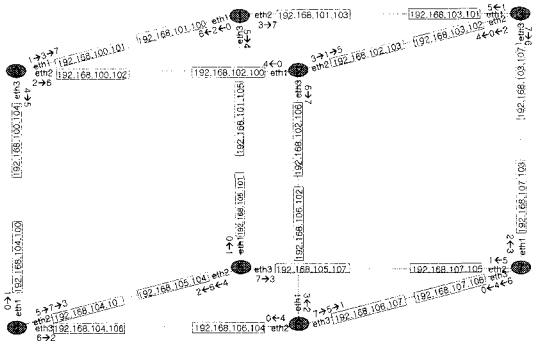

그림 5. [표 4]에 의한 하이퍼큐브 네트워크
Fig 5. Hypercube network based on Table 4

3.2.2.2 노드 위주의 IP 주소 설정 방안

링크 위주보다는 설정과 시각적 이해의 편의를 위해 net id를 노드 위주로 부여할 수 있다. 즉, 어느 특정 노드에 속한 모든 인터페이스에 모두 동일한 net id를 부여하는 방안으로, 이를테면 〈그림 3〉에서 node0에 장착된 3 개의 인터페이스에 a, b, c에 모두 동일한 net id를 부여하는 방식이다. 이는, 점대점 환경에서는 호스트가 아닌 인터페이스 혹은 디바이스 기반 라우팅 기능[10]이 있기 때문에 가능하다. 또한, TCP/IP의 노드간 직접적인 통신 선로는 동일한 net id를 가지는 노드 사이에서 설정되지만, 이러한 인터페이스 기반 라우팅 기능이 있기 때문에 링크 양쪽 인터페이스 IP 주소의 net id의 일치성 여부에는 관심을 두지 않는다. 즉, 〈그림 3〉에서 각각의 점대점 연결을 공유하는 a와 o, b와 x, c와 s, p와 q 등은 각각 동일한 net id를 가져야 하지만, 하이퍼큐브 네트워크 환경에서는 일치시키지 않아도 무방한 것이다. [표 4]와 〈그림 5〉에 노드 위주의 IP 주소부여 방안을 보였다.

표 4. 링크 위주의 IP 주소 설정 방안
Table 4. IP configuration based on link

노드[interface]	점대점 IP주소(xx.yy=192.168)	[interface]노드
node0[eth1]	xx.yy.100.101↔xx.yy.101.100	[eth1]node1
node0[eth2]	xx.yy.100.102↔xx.yy.102.100	[eth1]node2
node0[eth3]	xx.yy.100.103↔xx.yy.103.100	[eth1]node4
node1[eth2]	xx.yy.101.103↔xx.yy.103.101	[eth1]node3
node1[eth3]	xx.yy.101.105↔xx.yy.105.101	[eth1]node5
node2[eth2]	xx.yy.102.103↔xx.yy.103.102	[eth2]node3
node2[eth3]	xx.yy.102.106↔xx.yy.106.102	[eth1]node6
node3[eth3]	xx.yy.103.107↔xx.yy.107.103	[eth1]node7
node4[eth2]	xx.yy.104.105↔xx.yy.105.104	[eth2]node5
node4[eth3]	xx.yy.104.106↔xx.yy.106.104	[eth2]node6
node5[eth3]	xx.yy.105.107↔xx.yy.107.105	[eth2]node7
node6[eth3]	xx.yy.106.103↔xx.yy.107.106	[eth3]node7

3.2.3 통신경로 및 호스트 주소 설정

■ 통신경로 설계

하이퍼큐브 네트워크에서 어느 특정 노드를 기준으로 바로 인접한 노드에 대한 최단 통신경로는 명확하게 하나이다. 그러나 1 개 이상의 노드를 거쳐야 하는 목적 노드에 대한 최단 통신경로는 한 개 이상 존재한다. 이를테면, 〈그림 3〉에서 node0에서 node6으로 가는 가장 짧은 경로는 node2를 거치는 경로와 node4를 거치는 경로 등 2 개가 존재한다.

병렬처리를 위한 하이퍼큐브 네트워크에서는 통신 부하를 최대한 분산시키기 위하여 가는 방향과 오는 방향 각각에 대하여 서로 다른 통신경로를 사용하는 것이 바람직하다. 이를 위해서는 임의의 두 노드 간 통신경로를 하이퍼큐브의 입체적인 특성을 활용하여 상호 대칭적으로 설정하는 방안이 있다. 예를 들어 〈그림 3〉에서 node0→node1→node3→node7과 node7→node6→node4→node0은 입체적으로 상호 대칭 경로이다. 이 논문에서 제안한 거리 임의의 두 노드 사이의 대칭 경로를 〈그림 6〉에 보였다.

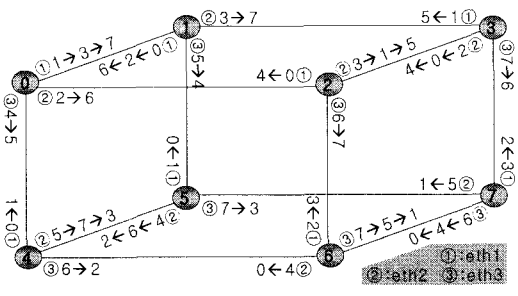

그림 6. 두 노드 사이의 경로
Fig 6. Paths between two nodes

■ 호스트 주소 설정

일반적으로 노드가 여러 개의 인터페이스를 가진 경우 각각의 인터페이스를 대변하는 3 개의 서로 다른 노드 id를 갖는 것이 보통이지만, 하이퍼큐브 네트워크에서는 노드별로 유일한 하나의 노드 id 만을 사용해야 하므로 전체적으로 노드 id는 통일하여 사용하되, 노드 id에 대응되는 IP 주소는 각 개별 노드 관점에서 주시되는 주소로 설정해야 한다. 예를 들어, 〈그림 3〉에서 node0의 주소는 node1의 관점에서 a이지만 node2의 관점에서는 b가 된다. 마찬가지로 node4의 관점에서 node0의 주소는 c이다.

거리가 2 이상인 경우에는 목적노드의 주소가 설계된 경로에 따라 결정된다. 이를테면, 〈그림 3〉에서 node0→node1→node3의 경로가 선택되었다면 node0에서 주시되는 node3의 주소는 q가 된다. [표 5], [표 6]에 링크 위주 IP주소 설정([표 3], 〈그림 4〉)와 노드 위주의 IP주소 설정([표 4], 〈그림 5〉) 환경하에서의 호스트 주소 설정("/etc/hosts")을 각각 보였다.

표 5. [표 3]에 의한 호스트 주소
Table 5. Host addresses based on Table 3

기 준	목적 노드의 IP 주소(서두 '192.168.' 생략)					
node0			node1	101.110	node2	102.120
	node3	123.132	node4	104.140	node5	145.154
	node6	126.162	node7	137.173		
node1	node0	101.101			node2	123.123
	node3	113.131	node4	145.145	node5	115.151
	node6	167.167	node7	137.173		
node2	node0	102.102	node1	101.110		
	node3	123.132	node4	104.140	node5	145.154
	node6	126.162	node7	167.176		
node3	node0	101.101	node1	113.113	node2	123.123
			node4	145.145	node5	157.157
	node6	167.167	node7	137.173		
node4	node0	104.104	node1	101.110	node2	126.126
	node3	123.132			node5	145.154
	node6	146.164	node7	167.176		
node5	node0	101.101	node1	115.115	node2	123.123
	node3	113.131	node4	145.145		
	node6	167.167	node7	157.175		
node6	node0	104.104	node1	101.110	node2	126.126
	node3	123.132	node4	146.146	node5	145.154
			node7	167.176		
node7	node0	104.104	node1	115.115	node2	123.123
	node3	137.137	node4	145.145	node5	157.157
	node6	167.167				

표 6. [표 4]에 의한 호스트 주소
Table 6. Host addresses based on Table 4

기 준	목적 노드의 IP 주소(서두 '192.168.' 생략)					
node0			node1	101.100	node2	102.100
	node3	103.101	node4	105.104	node5	105.104
	node6	106.102	node7	107.103		
node1	node0	100.101			node2	102.100
	node3	103.101	node4	104.105	node5	105.101
	node6	106.102	node7	107.103		
node2	node0	100.102	node1	101.103		
	node3	103.102	node4	104.100	node5	105.101
	node6	106.102	node7	107.106		
node3	node0	100.102	node1	101.103	node2	102.103
			node4	104.100	node5	105.101
	node6	106.107	node7	107.103		
node4	node0	100.104	node1	101.100	node2	102.106
	node3	103.107			node5	105.104
	node6	106.104	node7	107.105		
node5	node0	100.101	node1	101.105	node2	102.106
	node3	103.107	node4	104.105		
	node6	106.104	node7	107.105		
node6	node0	100.104	node1	101.105	node2	102.106
	node3	103.102	node4	104.106	node5	105.107
			node7	107.106		
node7	node0	100.104	node1	101.105	node2	102.103
	node3	103.107	node4	104.106	node5	105.107
	node6	106.107				

3.2.4 라우팅 테이블 설정

라우팅 테이블 설정은 이 논문에서 하이퍼큐브 네트워크를 구현하기 위한 가장 핵심적인 과정이다. IP 주소 설정 방안과 관계없이 〈그림 6〉에 설계된 통신경로를 얻기 위해서는 [표 7]과 같이 각 노드별로 최소한 4 가지 라우팅 정보가 필요하다. 예를 들면, node0에서는 인접 노드가 아닌 node3, node7, node6, node5에 다다를 수 있는 각각의 중간 경유 노드를 인접노드인 node1, node2, node4 중에서 하나를 선택해야 한다. 〈그림 6〉으로부터 node0에서 node7에 닿기 위해서는 인터페이스 eth1에 연결된 node1을 거쳐야 한다. 이를 〈node7:node1〉 형태로 표현하고, 이 때 node7은 목적노드, node1은 인접경유노드가 된다.

표 7. 〈그림 6〉을 위한 라우팅 정보
Table 7. Routing information for Fig 6

노 드	라 우 팅 정 보 [interface]
node0	<node3 : node1[eth1]>,<node7 : node1[eth1]>, <node6 : node2[eth2]>,<node5 : node4[eth3]>
node1	<node2 : node0[eth1]>,<node6 : node0[eth1]>, <node7 : node3[eth2]>,<node4 : node5[eth3]>
node2	<node4 : node0[eth1]>,<node1 : node3[eth2]>, <node5 : node3[eth2]>,<node7 : node6[eth3]>
node3	<node5 : node1[eth1]>,<node0 : node2[eth2]>, <node4 : node2[eth2]>,<node6 : node7[eth3]>
node4	<node1 : node0[eth1]>,<node7 : node5[eth2]>, <node3 : node5[eth2]>,<node2 : node6[eth3]>
node5	<node0 : node1[eth1]>,<node6 : node4[eth2]>, <node2 : node4[eth2]>,<node3 : node7[eth3]>
node6	<node3 : node2[eth1]>,<node0 : node4[eth2]>, <node5 : node7[eth3]>,<node1 : node7[eth3]>
node7	<node2 : node3[eth1]>,<node1 : node5[eth2]>, <node4 : node6[eth3]>,<node0 : node6[eth3]>

% <node_i:node_j>→node_i는 목적노드,node_j는 인접노드

4.2.4.1 링크 위주 IP 주소 환경에서의 라우팅 테이블 설정

■ 거리 2 이상인 경로들의 라우팅 테이블

〈목적노드:인접경유노드〉 형태의 라우팅 정보에서 목적노드와 인접경유노드의 호스트 주소는 기본적으로 [표 5]에 설정된 노드주소를 사용한다. 그런데, 순방향에서의 출발노드는 대칭(역방향) 경로에서는 목적노드가 되기 때문에 응답(ack)을 위한 라우팅 정보도 고려되어야 한다. 이를테면, 〈그림 6〉의 node0→node1→node3→node7 경로 중, node 0에서 요구되는 라우팅 정보는 〈node7:node1〉인데, 이 때 node7의 호스트 주소는 node0에서 주시되는 192.168.137.173이지만, 대칭경로 node7→node6→node4→node0에서 출발노드인 node7에 대한 응답패킷을 보낼 때의 node7의 주소는 node6에서 주시되는 192.168. 167.176이 된다. 따라서, 라우팅 정보 〈node7:node1〉을 위해〈192.168.137.173 : 192.168.101.110 〉와 〈192.168.167.176 : 192.168.101.110〉 등 2 개의 라우팅 테이블이 필요하다.

위 경우를 일반화시키면, 특정 노드의 라우팅 정보 관점에서 목적노드에 부여된 세 개의 인터페이스 주소 중 어느 것이라도 처리할 수 있도록 하기 위해서는 해당 라우팅 정보에 대하여 3 개의 라우팅 테이블이 필요하다. 즉, 바로 위의 예에서 node0의 라우팅 정보 〈node7:node1〉에 대하여 〈192.168.137.173 : 192.168.101.110〉, 〈 192.168.167.176 : 192.168.101.110〉, 〈192.168. 157.175 : 192.168.101.110〉 등 3 개의 라우팅 테이블을 설정해야 한다.

■ 길이 1인 경로들(인접노드)의 라우팅 테이블

IP 프로토콜에는 인터페이스가 여러 개인 경우 목적노드 주소의 net id와 일치하는 인터페이스를 자동으로 검색하여 전달하는 "ip forward"기능이 기본 기능으로 내장되어 있다. 그러나, 최근에는 보안상의 이유로 이 기능을 기본적으로 비활성화 시키고, 필요에 따라 선택적으로 활성화시킬 수 있도록 한다. 즉, 모든 인터페이스에 대한 라우팅 정보를 제공해야 한다는 것이다.

예를 들어, 〈그림 6〉의 node0는 주위에 node1, node5, node4 등 3 개의 인접노드를 가지고 있기 때문에 각각을 위해 〈node1, ①〉, 〈node5, ②〉, 〈node4, ③〉의 라우팅 정보가 필요하다. 이들 라우팅 정보에서 node1, node4, node5는 각각 3 개의 서로 다른 주소를 가지므로 라우팅 테이블 또한 각각에 대하여 3 개씩 설정해야 한다. 라우팅 정보 〈node1, ①〉의 경우, 〈192.168.113.113, ①〉, 〈192.168.115.115, ① 〉, 〈192.168.101.110, ①〉 등 3 개의 라우팅 테이블이 필요하다.

■ 라우팅 테이블 설정 스크립트

라우팅 테이블을 설정하는 명령어 route의 사용법은 시스템에 따라 조금 다를 수 있다. 〈그림 7〉에 이 논문에서 사용한 시스템([표 1] 참조)에서 node0를 위한 인터페이스 주소 및 라우팅 테이블 설정을 위한 스크립트를 보였는데, 나머지 node1~node7을 위한 설정 내용도 이와 유사하다.

```
/sbin/ifconfig eth1 192.168.101.101/24 up
/sbin/ifconfig eth2 192.168.102.102/24 up
/sbin/ifconfig eth3 192.168.104.104/24 up

/sbin/ip route add 192.168.101.110 dev eth1 #1
/sbin/ip route add 192.168.113.113 dev eth1 #1
/sbin/ip route add 192.168.115.115 dev eth1 #1

/sbin/ip route add 192.168.102.120 dev eth2 #2
/sbin/ip route add 192.168.126.126 dev eth2 #2
/sbin/ip route add 192.168.123.123 dev eth2 #2

/sbin/ip route add 192.168.104.140 dev eth3 #4
/sbin/ip route add 192.168.145.145 dev eth3 #4
/sbin/ip route add 192.168.146.146 dev eth3 #4

/sbin/ip route add 192.168.113.131 via 192.168.101.110 #1-3
/sbin/ip route add 192.168.123.132 via 192.168.101.110 #1-3
/sbin/ip route add 192.168.137.137 via 192.168.101.110 #1-3

/sbin/ip route add 192.168.126.162 via 192.168.102.120 #2-6
/sbin/ip route add 192.168.146.164 via 192.168.102.120 #2-6
/sbin/ip route add 192.168.167.167 via 192.168.102.120 #2-6

/sbin/ip route add 192.168.145.154 via 192.168.104.140 #4-5
/sbin/ip route add 192.168.115.151 via 192.168.104.140 #4-5
/sbin/ip route add 192.168.157.157 via 192.168.104.140 #4-5

/sbin/ip route add 192.168.137.173 via 192.168.101.110 #1-3-7
/sbin/ip route add 192.168.167.176 via 192.168.101.110 #1-3-7
/sbin/ip route add 192.168.157.175 via 192.168.101.110 #1-3-7
```

그림 7. 링크 위주의 IP 설정 환경에서 node0의 네트워크 설정 스크립트
Fig 7. Network configuration script for node0 in link based IP environment

3.2.4.2 노드 위주 IP 주소 환경에서의 라우팅 테이블 설정

링크 위주의 IP 주소 환경에서는 라우팅 테이블을 호스트(-host) 기반으로 설정할 수 밖에 없었으나, 노드 위주의 IP 주소 환경에서는 노드별로net id가 동일하므로 네트워크

(-net) 기반의 라우팅 테이블 설정이 가능하다. 〈그림 8〉에 node0를 위한 인터페이스 주소와 라우팅 테이블 설정을 위한 스크립트를 보였고, node1~node7을 위한 설정 내용도 이와 유사하다.

```
/sbin/ifconfig eth1 192.168.100.101/24 up
/sbin/ifconfig eth2 192.168.100.102/24 up
/sbin/ifconfig eth3 192.168.100.104/24 up

/sbin/ip route add 192.168.101.0/24 dev eth1 #1
/sbin/ip route add 192.168.102.0/24 dev eth2 #2
/sbin/ip route add 192.168.104.0/24 dev eth3 #4

/sbin/ip route add 192.168.103.0/24 via 192.168.101.100 #1-3
/sbin/ip route add 192.168.106.0/24 via 192.168.102.100 #2-6
/sbin/ip route add 192.168.105.0/24 via 192.168.104.100 #4-5

/sbin/ip route add 192.168.107.0/24 via 192.168.101.100 #1-3-7
```

그림 8. 노드 위주의 IP 설정 환경에서 node0의 네트워크 설정 스크립트
Fig 8. Network configuration script for node0 in node based IP environment

3.3 TCP/IP 기반 스타 네트워크의 구현

TCP/IP 프로토콜을 사용하는 스타 네트워크는 〈그림 1〉과 같이 LAN 스위치를 중심으로 8대의 PC를 간접 연결하고, 임의의 두 노드 간에 통신 경로 설정이 가능하도록 각 PC의 인터페이스에 동일한 net id를 가지는 IP주소를 설정한다. 이 논문에서는 실험의 편리성을 위해 각 PC에 네 개의 LAN 카드를 장착하고 그 중 한 개(eth0)는 스타 네트워크 구성에 사용하고, 나머지 세 개(eth1~eth3)는 하이퍼큐브 네트워크 구성에 사용하였다. 〔표 8〕에 스타 네트워크 구축을 위해 지정한 각 노드의 IP 주소를 보였다(nodei에서 하이퍼큐브용 nodei는 무의미함).

표 8. 스타 네트워크를 위한 네트워크 설정
Table 8. Network configuration for star network

노드(alias, interface)	인터페이스 IP 주소
node0(hpc0, eth0)	192.168.0.100
node1(hpc1, eth0)	192.168.0.101
node2(hpc2, eth0)	192.168.0.102
node3(hpc3, eth0)	192.168.0.103
node4(hpc4, eth0)	192.168.0.104
node5(hpc5, eth0)	192.168.0.105
node6(hpc6, eth0)	192.168.0.106
node7(hpc7, eth0)	192.168.0.107

IV. 구현 하이퍼큐브 네트워크 평가

구현 하이퍼큐브 네트워크의 평가를 위해서 우선, 〈그림 7〉, 〈그림 8〉에 제안된 두 가지 라우팅 설정 방안에 대한 기본 기능과 성능을 비교 한 후, 스타 네트워크 기반 클러스타와의 비교를 통해서 병렬처리 환경에서의 기능을 검증한다.

4.1 평가 도구

평가 항목으로 〔표 9〕와 같이 노드간 단순 통신, 브로드캐스트, 병렬처리 등 크게 3 분야로 분류하여 각 분야에 적합한 일반적인 도구나 전형적인 병렬처리 프로그램을 적용하였다. Ping과 netperf는 일반적인 TCP/ IP 진단 및 벤치마크 도구들이고, MPICH〔5〕는 잘 알려진 표준 병렬처리 라이브러리다.

표 9. 하이퍼큐브 네트워크 평가 도구
Table 9. Evaluation tools for hypercube network

검증분야	검증도구	
노드간 단순 통신	일반적인 도구	ping
		netperf
브로드캐스트	병렬처리 도구	MPICH(Send/Recv 구현)
		MPICH(Bcast 구현)
병렬처리		MPICH(Canon 알고리즘 구현)

〔표 9〕의 도구별 적용 방법은 아래와 같다.

■ ping
- 패킷 크기 : 64 Kbytes(라우팅에 의한 미세한 성능상의 차이점이 부각되기 위해서는 가급적 큰 패킷을 사용하는 것이 바람직하므로 최대 크기에 가까운 패킷 크기 사용)
- 측정 방법 : 특정 노드에서 보낸 패킷이 거리 1, 2, 3인 목적 노드를 거쳐 되돌아오는 데 소요된 시간을 10 회 측정한 다음, 그 결과의 평균을 Mbps 단위로 변환

■ netperf
- 패킷 크기 : 4 Kbytes(이더넷 MTU 크기를 충분히 벗어날 수 있는 크기 사용)
- 측정 방법 : 거리가 1, 2, 3인 임의의 두 노드간에 netperf 클라이언트와 서버의 10초간 실행 과정에서의 CPU 이용률 10회 측정한 다음, 그 결과의 평균을 Mbps 단위로 변환

■ MPI_Send/Recv
- 데이터 크기 : 200 Mbytes(충분이 많은 패킷들이 전송될 수 있을 정도)
- 측정 방법 : MPI(Massage Passing Interface)〔5〕를 사용하여 두 노드간에 데이터를 송수신하는 프로그램을 구현하고(〔부록 1〕 참조), 거리 1, 2, 3인 임의의 두 노

드 간 데이터 전송 시간을 10 회 측정한 다음, 그 결과의 평균을 Mbps 단위로 변환

■ MPI_Bcast

◦ 데이터 크기 : 데이터 크기에 따른 성능 특성 관찰을 겸하기 위해 세 개의 2차원 정수 배열500500, 600600, 700700을 사용

◦ 측정 방법 : MPI를 이용하여 구현한 프로그램으로([부록 2] 참조), 특정 노드에서 PC 클러스터에 참여하고 있는 다른 모든 노드에게 데이터를 브로드캐스트하는데 소요되는 시간을 여러 번 측정한 다음, 그 결과의 평균을 Mbps 단위로 변환

4.2 측정 및 비교 방법

링크 위주의 IP 설정과 노드 위주의 IP 설정 방법의 비교·평가를 위해 ping, netperf, MPI_Send/ Recv, MPI_Bcast에 의한 전송성능을 측정으로 하였고, 구현된 하이퍼큐브 네트워크의 기능적 무결성 검증을 위해서는 스타 네트워크 기반 PC클러스터에 MPI_Canon 을 동일하게 적용하여 처리된 결과를 하이퍼큐브 네트워크 기반 클러스터에서의 처리결과와 비교하였다.

도표 비교의 편리성을 위해 위의 측정 환경을 [표10]과 같이 분류하여 명명하였다.

표 10. 측정 클러스터 환경의 분류
Table 10. Classification of clusterevaluation environment

측정 클러스터 환경 분류		클러스터 명칭
하이퍼 큐브 네트워크	링크 위주 IP 주소([그림 7])	Hyper1
	노드 위주 IP 주소([그림 8])	Hyper2
스타 네트워크		Star

4.3 측정 결과 및 분석

4.3.1 노드간 단순 통신 기능 및 성능

Node0~node7 각각에서 나머지 7 개의 노드와의 ping 측정 결과 중, node0에서 측정된 결과를 〈그림 9〉에 보였는데, 이 결과로부터 Hyper1과 Hyper2는 모든 노드간 통신기능이 정상이고, 시간적 성능 측면에서 거의 동일함을 확인할 수 있다. 특히, 시간적 성능의 경우 운영체제의 측정 오차로 인하여 우열 관계가 일정하지 않아 두 이 실험에서는 우열을 판단할 수 없었다.

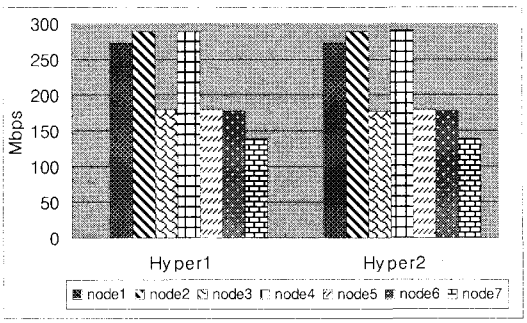

그림 9. node0에서 측정된 ping의 성능
Fig 9. Performance of ping at node0

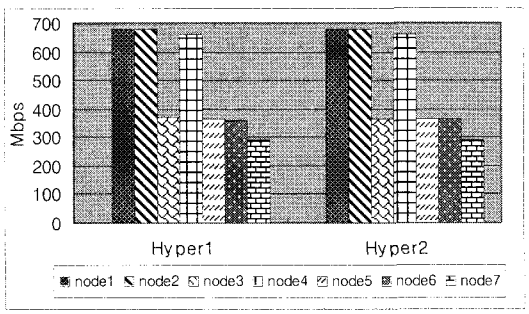

그림 10. node0에서 측정된 netperf의 성능
Fig 10. Performance of netperf at node0

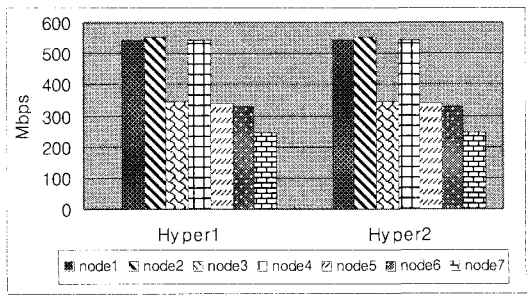

그림 11. node0에서 측정된 MPI_S/R의 성능
Fig 11. Performance of MPI_S/R at node0

〈그림 10〉에는 ping과 마찬가지 방법으로 측정된 netperf의 측정 결과 중, node0에서 측정된 결과를 보였다. 이 결과에서도 ping과 동일한 결론을 얻을 수 있다.

병렬처리 도구인 MPI_Send/Recv에 의한 노드간 단순 통신 성능 측정결과를 〈그림 11〉에 보였는데, 이 결과에서도 ping 및 netperf와 동일한 결론을 보이고 있다.

4.3.2 브로드캐스팅 기능 및 성능

병렬처리 도구인 MPI_Bcast를 사용하여 node 0~node7 각각에서 나머지 7 개의 노드로의 브로드캐스트를 하는 성능 측정 결과 중, node0에서 측정된 결과를 〈그림 12〉에 보였다. 이 결과에서도 Hper1과 Hyper2의 양쪽 모두 기능은 검증되었으나, 시간적 성능 측면에서는 뚜렷한 우열을 가늠할 수는 없다.

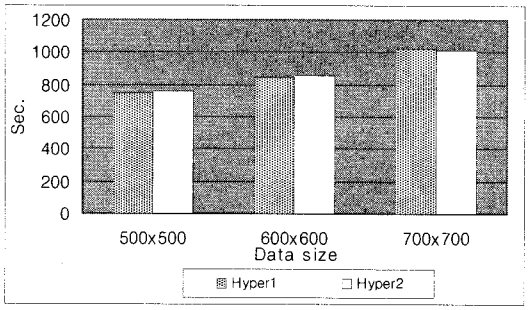

그림 12. node0에서 측정된 MPI_Bcast의 성능
Fig 12. Performance of MPI_Bcast at node0

4.3.3 병렬처리 기능

마지막으로, 구현된 하이퍼큐브 네트워크 기반 PC클러스터의 병렬처리 기능을 검증하기 위하여 병렬처리 도구인 MPI_Canon을 [표 10]의 세 가지 PC 클러스터에서 실행하여 동일한 처리 결과를 내고 종료할 때까지 측정된 시간 〈그림 13〉에 보였다. 이 실험에서의 관심은, 스타 네트워크 기반 클러스터와 하이퍼큐브 네트워크 기반 클러스터 사이의 시간적 성능 비교에 있지 않고, 양쪽의 결과를 비교함으로써 구현된 하이퍼큐브 네트워크가 병렬처리를 지원하는지의 기능적 관점에 있다.

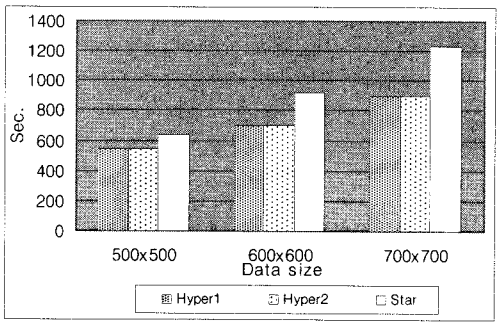

그림 13. node0에서 측정된 MPI_Cannon의 성능
Fig 13. Performance of MPI_Cannon at node0

〈그림 13〉으로부터 하이퍼큐브 네트워크 기반 클러스터가 스타네트워크 기반 클러스터와 동일하게 병렬처리를 완벽하게 지원함을 확인할 수 있다. 이 그림에서, 하이퍼큐브 네트워크 기반 클러스터의 시간적 성능이 우수한 것으로 나타나 있으나 이는 스타 네트워크에 사용된 스위치 및 LAN 카드에 대한 객관적인 성능 분석이 이루어지지 않았으므로 절대적인 의미를 부여할 수는 없다.

V. 결론

일반적으로 클러스터는 Myrinet, GigEther net, Quadrics 등 별도의 전용 스위치 장비를 사용하여 스타 네트워크, 하이퍼큐브 네트워크, 크로스바 네트워크 등 고유의 토폴로지 위상으로 구축되어 가격이 비싸다. 또한 전용 시스템의 경우, 장비에 대한 의존도가 높아지게 되어 장비에 문제가 발생할 때에는 클러스터 기능에 직접적인 영향을 미치게 된다.

이 논문에서는 TCP/IP 환경에서 별도의 스위치를 사용하지 않고 평범한 8 대의 PC들을 직접 연결하여 하이퍼큐브 네트워크 기반 PC 클러스터를 구현하기 위한 설정 방안으로, 링크 위주의 IP 설정 방법(Hyper1)과 노드 위주의 IP 설정 방법(Hyper2)을 제안하고, 그 각각의 기능 및 성능을 다양한 도구를 사용하여 검증하였다. 그 결과, 두 방법 사이의 시간적 성능에 관해서는 우열을 판가름할 수는 없었으나, Hyper2 방법의 라우팅 테이블의 크기가 Hyper1에 비해 1/3에 지나지 않기 때문에 Hyper1의 성능이 미세하지만 우수하다고 결론지어도 전혀 무리가 없다. 뿐만 아니라, Hyper2 방법이 노드별로 동일한 net id를 부여하기 때문에 하이퍼큐브에 대한 입체적인 가독성 측면에서도 우수하다.

또한, 구현된 하이퍼큐브 네트워크 기반 클러스터의 기능은 스타네트워크 기반 클러스터와 동일한 병렬처리 환경을 지원한다는 사실로 검증되었다.

이 논문의 의의는 별도의 네트워크장비를 사용하지 않고, 주위의 일상적인 PC를 직접 이용하여 실질적인 하이퍼큐브 네트워크 기반 병렬처리 환경을 보다 효율적으로 구축하는 방안을 제안했다는 점에 있다고 본다.

참고문헌

[1] http://www.cpubenchmark.net, 2007

[2] http://www.top500.org/list/2007/11/200, 2007

[3] Heieh, J., Leng, T., Mashayekhi, V. and Rooholamini, R., "Architectural and Performance Evaluation of GigaNet and Myrinet Interconnects on Clusters of Small-Scale SMP Servers", Proc. Supercomputing Conference, Dallas, USA, 2000.

[4] 김영태, 이용희, 최준태, 오재호, "초고속 네트워크를 이용한 PC 클러스터의 구현과 성능 평가", 정보과학회논문지:시스템 및 이론, 제29권 2호, pp. 57-64, Feb. 2002.

[5] Pacheco P., Parallel Programming with MPI, San Francisco, CA: Morgan Kaufmann, 1997.

[6] Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K. and Walker, D. W., Solving Problems On Concurrent Processors Volume I: General Techniques and Regular Problem, Engelwood Cliffs: Prentice-Hall 1998.

[7] 명훈주, 김성천, "하이퍼큐브 컴퓨터에서 효과적인 오류 허용 다중 전송기법", 정보과학회논문지:시스템 및 이론, 제30권 제 56호, pp. 273-279, Jun. 2003.

[8] 김종석, 오은숙, 이형옥, "하이퍼-스타 연결망의 위상적 성질과 방송 알고리즘", 한국정보처리학회 논문지A, 제 11-A권 제5호, pp. 341-346, Oct. 2004.

[9] W. Rechard Stevens, UNIX Network Programming Volume 1, Prentice Hall PTR, pp. 883-962. 1998

[10] Digital UNIX, Reference Pages Sections 8 and 1m:Syatem Administration Commands, Volume 2:route, pp. 1-134~ 136, Digital Press, 1996

[11] Kirch, O. and Dawson, T., Linux Network Administrator's Guide,Sebastopol, CA: O'reilly, 2000.

[12] 김재열, 강동재, 김수영, 차규일, "리눅스 Bonding 드라이버의 성능분석", 한국통신학회, KNOM Review Vol.8, No.1, pp. 57-75, Aug. 2005.

[13] Hewlett-Packard Company, Information Networks Division : A Network Performance Benchmark, 1995.

[14] Kim, Y., Towards a Fair Comparison of Parallel Machines, Journal of KISS(A): Computer Systems and Theory (정보과학회논문지(A)), Vol. 26, No. 1, pp. 43-52, Jan. 1999.

[15] Cannon, L. E., "A cellular computer to implement the Kalman Filter Algorithm", Ph.D. thesis, Montana State University, 1969.

부록 1] MPI_Send/Recv 구현 코드

```
#include <stdlib.h>
#include <mpi.h>
#include <math.h>
#define MAX 200000000
#define fl(i,j) *(fl+(MAX*i)+j)
int main(int argc, char *argv[])
{
    int my_rank,p,tag=0;
    char *fl;
    int start,stop,i,j;
    MPI_Status status;
    fl = (char *) malloc(sizeof(char)*MAX);
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    for(i=0; i<MAX;i++)
        fl[i] = 'a';
    /* check time */
    if(my_rank == 0) start = milliclock();
    if(my_rank == 0) {
        for(i=1; i<p;i++)

      MPI_Send(fl,MAX,MPI_CHAR,i,tag,MPI_COMM_WORLD);
    } else
MPI_Recv(fl,MAX,MPI_CHAR,0,tag,MPI_COMM_WORLD,
                                        &status);
    /* check time */
    if(my_rank == 0) {
        stop = milliclock();
        printf("%d\n", stop - start);
    }
    free(fl);
    /* parallel program end */
    MPI_Finalize();
}
#include <sys/time.h>
#include <unistd.h>
milliclock()
{
    struct timeval tb;
    struct timezone tzp;
    int isec, usec, msecs;
    gettimeofday(&tb,&tzp);
    isec = tb.tv_sec;
    usec = tb.tv_usec;
    msecs = 1000 * isec + usec/1000;
    return(msecs);
}
```

[부록 2] MPI_Bcast 구현 코드

```
#include <stdlib.h>
#include "mpi.h"
#include <math.h>
#define MAX 200000000
#define TIMES 1
#define fl(i,j) *(fl+(MAX*i)+j)
int main(int argc, char *argv[])
{
    int my_rank,p,tag=0;
    char *fl;
    int start,stop,i,j;
    MPI_Status status;
    fl = (char *) malloc(sizeof(char)*MAX);
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    for(i=0; i<MAX;i++)
        fl[i] = 'a';
    /* check time */
    if(my_rank == 0) start = milliclock();
    start = milliclock();
    for(i=0; i<TIMES; i++)
        MPI_Bcast(fl,MAX,MPI_CHAR,0,MPI_COMM_WORLD);
    /* check time */
```

```
    stop = milliclock();
    printf("my_rank : %d[%d]\n", my_rank,stop - start);
    if(my_rank == 0) {
        stop = milliclock();
        printf("%d\n", stop - start);
}
 free(fl);
 /* parallel program end */
 MPI_Finalize();
}
```

[부록 3] MPI_Cannon 구현 코드

```
#include <stdlib.h>
#include <mpi.h>
#include <math.h>
#define MAX 500
#define LOOP_TIMES 100
#define A(i,j) *(A+(i*MAX)+j)
#define B(i,j) *(B+(i*MAX)+j)
#define C(i,j) *(C+(i*MAX)+j)
#define local_a(i,j,n) *(local_a+(i*n)+j)
#define local_b(i,j,n) *(local_b+(i*n)+j)
#define local_c(i,j,n) *(local_c+(i*n)+j)
#define temp_a(i,j,n) *(temp_a +(i*n)+j)
#define temp_b(i,j,n) *(temp_b +(i*n)+j)
#define temp_matrix(i,j,n) *(temp_matrix+(i*n)+j)
int main(int argc, char *argv[])
{
      float *A, *B, *C, *local_a, *local_b, *local_c,
            *temp_matrix,*temp_a,*temp_b;
      int i,j,k,times, start_num,
      int dest_b,source_b, dest_a, source_a;
      int p, q, my_row, my_col, my_rank;
      int n_bar, start,stop, loop;
      MPI_Comm my_row_comm, my_col_comm;
      MPI_Status status;
      A = (float *)malloc(sizeof(float)*(MAX*MAX));
      B = (float *)malloc(sizeof(float)*(MAX*MAX));
      C = (float *)malloc(sizeof(float)*(MAX*MAX));
      local_a = (float *)malloc(sizeof(float)*(MAX*MAX));
      local_b = (float *)malloc(sizeof(float)*(MAX*MAX));
      local_c = (float *)malloc(sizeof(float)*(MAX*MAX));
      temp_matrix = (float *)malloc(sizeof(float)*(MAX*MAX));
      temp_a = (float *)malloc(sizeof(float)*(MAX*MAX));
      temp_b = (float *)malloc(sizeof(float)*(MAX*MAX));
      /* generate the input vectors */
      for(i=0; i<MAX; i++) {
          for(j=0; j<MAX; j++) {
           A(i,j) = (float)rand()/(float)RAND_MAX;
           B(i,j) = (float)rand()/(float)RAND_MAX;
       }
      }
      /* start parallel program */
      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &p);
      MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
      /* check time */
      if(my_rank == 0) start = milliclock();
      for(loop=0; loop<LOOP_TIMES; loop++) {
          q = (int)sqrt(p);
          my_col = my_rank % q;
          my_row = my_rank / q;
          n_bar = MAX/(int)sqrt(p);
          /* compute start position */
          start_num = ((MAX*MAX)/q) * (my_rank / q) +
                                          ( MAX / q * (my_rank%q));
           /* set local_a */
           for(i=0; i<n_bar; i++)
             for(j=0; j<n_bar; j++)
               local_a(i,j,n_bar) =*(A+start_num+(i*MAX)+j);
           /* set local_b */
           for(i=0; i<n_bar; i++)
             for(j=0; j<n_bar; j++)
               local_b(i,j,n_bar) =*(B+start_num+(i*MAX)+j);
           /* grind A */
           for(i=1; i<q+1; i++) {
             if(i-1 == (my_rank / q)) {
               if((my_rank%q)-i < 0) dest_a = my_rank+(q-i);
               else dest_a = my_rank - i;
               if(my_rank%q + i < q) source_a = my_rank + i;
               else source_a = my_rank+i-q;
             }
           }
           /* grind B */
           for(i=1; i<q+1; i++) {
             if(i-1 == (my_rank %q)) {
             if((my_rank/q)-i < 0)
               dest_b = q*q + my_rank - (i*q);
             else dest_b = my_rank - (i*q);
             if(my_rank + i*q < (q*q))
               source_b = my_rank + i*q;
             else source_b = my_rank - (q-i)*q;
           }
           }
           /* send A and B(grinding... ) */
           MPI_Send(local_a, n_bar*n_bar, MPI_FLOAT, dest_a,
                                0, MPI_COMM_WORLD);
           MPI_Recv(temp_a, n_bar*n_bar, MPI_FLOAT, source_a,
                                0, MPI_COMM_WORLD, &status);
           MPI_Send(local_b, n_bar*n_bar, MPI_FLOAT, dest_b,
                                0, MPI_COMM_WORLD);
           MPI_Recv(temp_b, n_bar*n_bar, MPI_FLOAT, source_b,
                                0, MPI_COMM_WORLD, &status);
           MPI_Barrier(MPI_COMM_WORLD);
           for(i=0; i<n_bar; i++)
             for(j=0; j<n_bar; j++) {
               local_a(i,j,n_bar) = temp_a(i,j,n_bar);
               local_b(i,j,n_bar) = temp_b(i,j,n_bar);
             }
             /* compute
           Diagonal of A shifted to rightmost column
           Diagonal of B shifted to bottom row
           */
           if((my_rank%q)-1 < 0)  dest_a = my_rank + q-1;
           else      dest_a = my_rank - 1;
           if((my_rank%q)+1 >= q) source_a = my_rank - (q-1);
           else      source_a = my_rank + 1;
           if((my_rank/q) == 0) dest_b = (q-1) * q + my_rank;
           else          dest_b = my_rank - q;
           if((my_rank/q)+1 >= q) source_b = my_rank - (q-1)* q;
           else      source_b = my_rank + q;
           /* compute algorithm (cannon's) */
           for(times=0; times<q; times++) {
             /* C += A + B */
             for(i=0; i<n_bar; i++)
               for(j=0; j<n_bar; j++)
                 for(k=0; k<n_bar; k++)
                   local_c(i,j,n_bar) +=
                     local_a(i,k,n_bar) * local_b(k,j,n_bar);
             /* shifted */
             MPI_Send(local_a, n_bar*n_bar, MPI_FLOAT,
                      dest_a, 0, MPI_COMM_WORLD);
             MPI_Recv(temp_a, n_bar*n_bar, MPI_FLOAT,
                      source_a, 0, MPI_COMM_WORLD, &status);
             MPI_Send(local_b, n_bar*n_bar, MPI_FLOAT,
                      dest_b, 0, MPI_COMM_WORLD);
             MPI_Recv(temp_b, n_bar*n_bar, MPI_FLOAT,
                 source_b, 0, MPI_COMM_WORLD, &status);
             MPI_Barrier(MPI_COMM_WORLD);
             for(i=0; i<n_bar; i++)
               for(j=0; j<n_bar; j++) {
                 local_a(i,j,n_bar) = temp_a(i,j,n_bar);
                 local_b(i,j,n_bar) = temp_b(i,j,n_bar);
               }
           }
           /* send local_c array to 0 rank */
           MPI_Send(local_c, n_bar*n_bar, MPI_FLOAT, 0, 0,
                                                MPI_COMM_WORLD);
           if(my_rank == 0) {
                for(i=0; i<p; i++) {
                   MPI_Recv(temp_matrix, n_bar*n_bar, MPI_FLOAT,
                                i, 0, MPI_COMM_WORLD, &status);
                   /* compute start position */
                   start_num = ((MAX*MAX)/q) * (i / q) +
                                                (MAX / q * (i%q));
                   for(j=0; j<n_bar; j++)
                     for(k=0; k<n_bar; k++)
                         *(C+(start_num + j*MAX+k)) =
                                                temp_matrix(j,k,n_bar);
                }
           }
           MPI_Barrier(MPI_COMM_WORLD);
      } /* end LOOP_TIMES */
      if(my_rank==0) {
            /* check time */
           stop = milliclock();
         printf("%dWn",  stop - start);
      }
      /* parallel program end */
      MPI_Finalize();
}
```

저 자 소 개

이 형 봉 (hblee@kangnung.ac.kr)

1984년 2월 서울대학교 계산통계학과 학사.
1986년 2월 서울대학교 계산통계학(전산과학)과 석사.
2002년 2월 강원대학교 컴퓨터과학과 박사.
1986년 3월~1993년 12월 LG전자 컴퓨터연구소 선임연구원.
1994년 2월~ 1999년 2월 한국디지탈㈜ 책임컨설턴트.
1999년 3월~2004년 2월 호남대학교 조교수.
2004년 3월~현재 강릉대학교 부교수.
관심분야 : 임베디드 시스템, 센서네트워크, 데이터 마이닝 알고리즘

홍 준 표 (newsam202@naver.com)

1998년 3월 ~ 2004년 3월 강릉대학교 컴퓨터공학과 졸업(학사).
2004년 9월~2007년 8년 강릉대학교 컴퓨터공학과 졸업(석사).
2007년 5월 2007년 춘계 정보 처리학회 우수논문상 수상.
2007년 10월~현재 재단법인 사회서비스관리센터 근무.
관심분야 : 초고속 컴퓨팅

김 영 태 (ytkim@kangnung.ac.kr)

1986년 연세대학교 수학과 학사.
1992년 미국 Iowa State Univ. Computer Science, M.S.
1996년 미국 Iowa State Univ. Computer Science, Ph.D.
1998년~현재 강릉대학교 컴퓨터공학과 교수.
연구분야 : 병렬처리, 초고속 컴퓨팅