Journal of Korea Multimedia Society Vol. 7, No. 10, October 2004(pp. 1400-1411)

확장과 동적재구성 가능한 클러스터기반의 인터넷서비스 시스템

김동근[†], 박세명[††]

요 약

오늘날 인터넷의 대중화로 부하가 많은 웹서비스를 제공하는 시스템들은 클러스터기반으로 이동하는 추세이다. 그러나, 기존의 서버 시스템들은 특정 목적을 위한 전용 클러스터 구조로 각 서비스를 위한 클러스터 자원을 각각 보유하고 있으므로 자원의 이용이 효율적이지 못한 문제를 가진다. 본 연구에서는 클러스터상의 자원을 공유하여 부하에 따라 처리 자원을 동적으로 재구성하는 응용서비스 플랫폼을 제안하였다. 제안된 응용서비스 플랫폼은 특정 응용서비스를 제공하는 전위응용서버와 전위응용서버 고부하시 작업을 분담, 처리하는 후위서버군을 기반으로 전위응용서버에서 응용서비스 요청을 처리 또는 분배하는 서비스관리자와 후위서버에서 응용서비스에 대한 요청을 처리하는 작업처리자, 그리고 부하에 따라 후위서버에 작업처리자를 생성 및 제거하는 부하관리자로 구성된다. 클러스터된 후위서버군의 효율적인 관리를 위해 PVM을 이용하였다. 구현된 시스템은 기존의 단일서버 시스템에 비해 안정적인 동작을 보이며, 필요한 자원을 동적으로 할당, 시스템을 재구성함으로써 부하의 변화에 보다 능동적으로 대처할 수 있음을 확인하였다.

Scalable and Dynamically Reconfigurable Internet Service System Based on Clustered System

Dong Keun Kim[†], Se Myung Park[††]

ABSTRACT

Recently, explosion of internet user requires fundamental changes on the architecture of Web service system, from single server system to clustered server system, in parallel with the effort for improving the scalability of the single internet server system. But current cluster-based server systems are dedicated to the single application, for example, One-IP server system. One-IP server system has a clustered computing node with the same function and tries to distribute each request based on the IP to the clustered node evenly. In this paper, we implemented the more useful application service platform. It works on shared clustered server(back-end server) with an application server(front-end server) for a particular service. An application server provides a particular service at a low load by itself, but as the load increases, it reconfigures itself with one or more available server from the shared cluster and distributes the load on selected server evenly. We used PVM for an effective management of the clustered server. We found the implemented application service platform provides more stable and scalable operation characteristics and has remarkable performance improvement on the dynamic load changes.

Key words: Cluster(클러스터), Scalability(확장성), One-IP System(단일아이피시스템), PVM(피브엠), Load Balance(부하균형화)

※ 교신저자(Corresponding Author) : 박세명, 주소 : 경남 김해시 어방동(612-749), 전화 : 055)320-3276, FAX : 055) 322-3107, E-mail : smpark@cs.inje.ac.kr
접수일 : 2004년 3월 26일, 완료일 : 2004년 7월 27일

[†] 준회원, 인제대학교 전산학과
(E-mail : dkkim404@hotmail.com)
[††] 정회원, 인제대학교컴퓨터공학부 교수
※ 본 연구는 2002년 인제장학재단 연구비지원에 의한 것임

1. 서 론

인터넷의 보급은 단순히 데이터의 공유단계를 넘어 이제 지리적으로 분산되어 있는 처리자원의 공유를 통한 처리자원의 지역적 한계를 극복할 수 있는 계기를 제공하였다. 지역적으로 산재한 자원의 활용을 위해 생성된 클러스터링의 개념의 확장으로 메타컴퓨팅과 그리드 컴퓨팅[1,2]이라는 개념을 도출하기에 이르렀으며, 근자에 와서는 클러스터 및 그리드 컴퓨팅을 위한 기반구축과 더불어 이러한 환경에 적합한 다양한 특수목적의 응용의 개발에 관심이 집중되고 있다. 국내에서도 2001년 말에 Grid 관련 연구그룹이 형성되어 연구가 이루어지고 있으며 다양한 응용이 개발되고 있다.[3] 응용 개발 환경으로는 MPI[4]나 PVM(Parallel Virtual Machine)[5] 그리고 Globus[6]기반의 개발도구인 MPICH[7]등을 들 수 있다.

한편, 현재의 모든 인터넷 응용서비스들에 대한 요구는 부하에 무관하게 균일한 양질의 서비스제공과 서비스의 안정성 및 신뢰성 제공으로 집중되고 있으며, 이러한 요구는 크게 동시 접속자 수를 10,000명까지 확장하고자하는 서버 구성기술 개선(C10K Problem[8])에 관한 연구와 서버의 부하를 클러스터상의 처리기에 분산시키고자 하는 One-IP 서버구성기술[9]의 두 방향으로 수용되고 있다.

C10K Problem은 단일서버기반으로 많은 수의 동시 접속자를 허용하기 위한 전략들인 반면, 클러스터기반은 네트워크로 연결된 자원을 사용할 수 있으므로 두 기술이 상호보완작업을 거쳐 궁극적으로는 기존의 단일서버기반의 응용서비스들이 클러스터기반으로 이전될 것으로 보여진다. 이러한 변화는 기존의 모든 응용서버 시스템의 전반에 걸쳐 적용될 수 있을 것이다. 비록 클러스터링의 개념이 도입되기 전이긴 하지만 유휴자원의 관리를 통해 자원을 효율적으로 이용하고자 시도한 Network Of Workstation (NOW) at UC Berkeley[10], 클러스터된 자원을 이용한 인터넷 서비스 시스템[11], 그리고 전용 클러스터 자원을 활용하는 SAN(System Area Network [12], Storage Area Network[13])과 분산저장장치(Network Attached Storage Devices(NASD) at CMU[10]등의 기술이 개발되었다.

그러나 최근 제안된 One-IP기반의 응용서비스 시스템들은 유휴 자원들을 하나의 응용서비스를 제공하기 위한 목적으로 클러스터링하고, 동일한 응용서비스를 클러스터된 후위서버(back-end server)마다 중복 설치하고, 요청을 분배 처리하는 구조를 가지고 있다. 물론 근자에 와서는 특히 고급 컴퓨팅 자원의 가용성 증대 및 통신 속도의 급격한 개선에 따라 값싼 비용으로 손쉽게 고성능 컴퓨팅 클러스터를 구축할 수 있게 되었으나 클러스터된 자원 관리의 효율성 및 다양한 서비스의 제공을 위해서는 클러스터된 자원을 공유할 수 있는 응용서버 구성기술에 대한 연구 및 개발이 필요하다. 근자에는 Globus기반의 특수목적의 슈퍼컴퓨터를 개발하고자하는 시도도 보이고 있으나 Globus의 경우 개발목적인 WAN기반의 자원공유를 위해 개발되었으므로 Globus가 지역내의 클러스터자원의 활용을 위한 응용분야에 적합한지에 대한 평가도 진행되어야 할 것으로 보인다.

따라서 본 논문에서는 유휴 컴퓨팅 자원으로 구성된 PVM기반의 클러스터를 이용하여 확장성있는 응용서비스를 제공할 수 있는 응용서비스 플랫폼을 설계, 구현하고, 성능을 평가하였다. 제안된 시스템은 응용서비스별로 서비스 요청 부하에 따라 클러스터상의 공유된 처리자원을 동적으로 할당하는 서비스 시스템의 동적 재구성을 지원하여 보다 안정적인 응답시간을 보장할 뿐 아니라, 이를 바탕으로 다양한 응용서비스를 보다 효과적으로 설치 운영할 수 있도록 일관된 인터페이스를 지원한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 구현한 응용서비스 플랫폼에 대한 전반적인 동작에 대해 기술하고, 3장에서는 구현된 시스템에 대한 구현과정을 설명한다. 그리고 4장에서는 시스템 성능 평가, 5장에서 결론 및 향후과제를 기술한다.

2. 응용서비스 플랫폼 설계

2.1 동작 개요

본 논문에서는 공유된 클러스터 자원을 이용하게 함으로써 다양한 응용서비스들이 클러스터 자원의 공유를 기반으로 확장성있는 서비스를 제공할 수 있는 응용서비스 플랫폼을 제안하였으며 제안된 응용서비스 플랫폼의 동작은 다음과 같다.

그림 1은 클러스터 자원을 동적으로 사용하여 각 응용서비스를 제공해주는 응용서비스 플랫폼의 동작 개요를 도식화한 것이다. 그림에서 Service A.

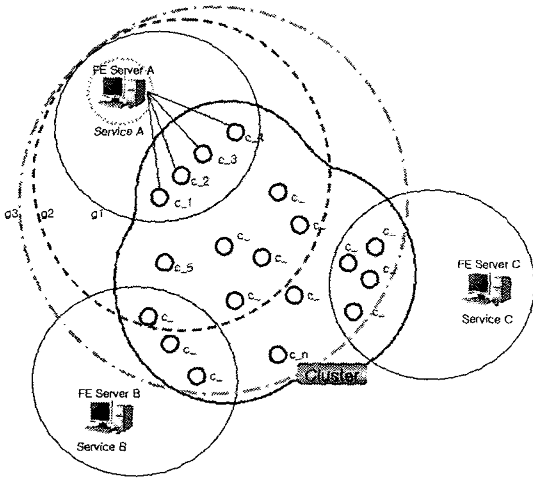

그림 1. 시스템 동작 개요

Service B, 그리고 Service C는 각 전위응용서버(FE Server)에서 제공되는 고유의 응용서비스로써 클라이언트의 서비스 요청은 저부하인 경우 해당 서비스의 전위응용서버에서 직접 처리된다. c_1, c_2, …, c_n 은 클러스터된 후위서버군으로 전위응용서버들에게 공유된다. 즉 각각의 응용서비스(Service A, Service B, 그리고 Service C) 요청은 해당 전위응용서버의 부하에 따라 해당 전위응용서버에서 직접 처리되거나, 공유된 클러스터의 후위서버를 할당받아 처리된다. 물론 부하의 저하로 더 이상 후위서버가 필요치 않으면 해당 후위서버 자원은 모든 서비스에서 사용 가능한 상태로 환원된다.

그림에 따라 설명하면, Service A를 제공하고 있는 전위응용서버 A는 현재 4개의 공유 클러스터 자원(g1)을 사용하여 Service A를 제공하고 있다. 부하가 증가하면 더 많은 클러스터 자원을 사용할 수 있고(g1→g2), 부하가 계속 증가하면 할당받는 클러스터 자원이 점차 증가하게 된다.(g2→g3). 또한, 부하가 계속 감소하면 사용하고 있는 4개의 클러스터 자원까지도 반환되고(g3→g2→g1), 결국에는 전위응용서버가 직접 해당 응용서비스를 제공하게 된다.

본 연구에서는 특정 응용서비스를 제공하는 전위응용서버(Front-end Application Server)와 전위응용서버의 고부하 시 작업을 분담, 처리하는 클러스터된 후위서버(Back-end Server)군을 기반으로 3개의 데몬, 즉 전위응용서버에서 응용서비스 요청을 처리 또는 분배하는 서비스관리자, 후위서버에서 응용서비스에 대한 요청을 처리하는 작업처리자, 그리고 부하에 따라 후위서버에 작업처리자를 생성 및 제거하는 부하관리자로 구성하였다.

2.2 시스템 구성 요소 및 기능

본 연구에서는 전 절에서 설명된 동작을 구현하기 위해 전위응용서버에 서비스관리자와 부하관리자, 그리고 후위서버에 작업처리자를 구현하였으며, 구현된 시스템의 구성도와 각 서버별 관리자들간의 연관성은 그림 2와 같다.

구현된 시스템은 크게 전위응용서버와 클러스터된 후위서버군으로 구성된다. 전위응용서버는 인터넷을 통해 클라이언트의 요청을 받아들이고, 처리 결과를 다시 해당 클라이언트에게 반환한다. 요청 처리를 위해 전위응용서버는 자신의 부하정도에 따라 클러스터 상에 공유된 후위서버군 중에서 유휴자원을 할당받아 요청 처리에 사용할 수 있다. 본 논문에서 구현한 각 관리자의 기능에 대한 간단한 설명은 아래와 같다.

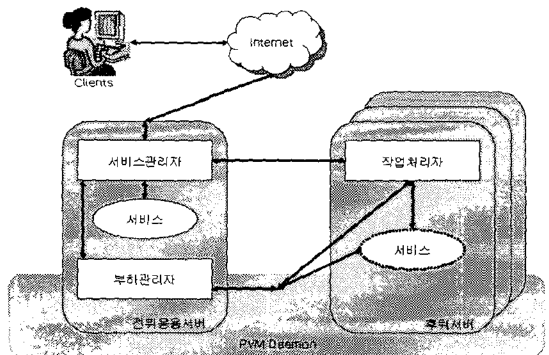

그림 2. 구현된 시스템 구성도

2.2.1 전위응용서버의 서비스관리자

전위응용서버의 서비스관리자는 클라이언트로부터 요청을 받으면, 자신의 서비스(저부하일 때) 혹은 해당 노드의 작업처리자(고부하일 때)에게 요청을 전달한다. 그리고 반환된 요청 처리 결과를 클라이언트에게 전달한다.

2.2.2 부하관리자

부하관리자는 전위응용서버에만 있으며, 전위응용서버의 부하와 사용 가능한 후위서버의 부하를 모니터링한다. 전위응용서버의 부하가 적정수준 이상

이 되면, 부하관리자는 후위서버중 저부하인 노드를 선택하고, 선택된 노드는 서비스관리자의 child_list에 등록된다. 이때, 선택된 노드의 작업처리자와 전위응용서버의 서비스와 동일한 응용서비스는 PVM 데몬을 통해 활성화된다. 그리고 리스트에 등록된 노드를 모니터하고 적정 수준 이상이 되거나, 그 이하로 떨어지면 child_list는 갱신된다. 여기에서 child_list는 서비스관리자가 요청을 자체적으로 처리할 것인지 클러스터 자원을 이용하여 처리할 것인지 결정기준이 된다. 즉, 서비스관리자는 child_list에 등록된 노드가 없으면 지역에 있는 서비스에게 요청을 전달하고, child_list에 등록된 노드가 있으면 해당 노드의 작업처리자에게 요청이 전달한다.

2.2.3 후위서버의 작업처리자

후위서버의 작업처리자는 전위응용서버로부터 받은 요청을 자신의 서비스들 중, 이미 활성화된 서비스에게 요청을 전달하고, 처리 결과를 받아서 전위응용서버에게 전달하는 기능을 수행하며, 부하관리자의 요청에 따라 현재 처리중인 요청의 수를 반환한다.

2.2.4 서비스

서비스는 클라이언트에게 실질적으로 제공되는 서비스로서 요청을 처리하고, 처리 결과는 작업관리자를 통해 또는 직접 서비스관리자에게 전달된다. 본 연구에서는 우선적으로 기본적인 웹서비스 기능만 갖는 HTTP-WebServer로 실험하였다. HTTP-WebServer를 이용한 실험에서는 반환되는 문서의 크기를 변경시키거나 문서 반환 시 지연시간을 설정함으로써 요청 당 처리시간의 변화에 따른 시스템의 성능을 평가하였다. 즉 실험시 문서의 크기를 변경함으로써 발생하는 통신비용이 전체 시스템에 끼치는 영향을 평가할 수 있을 뿐 아니라, 요청 처리 시 서비스에서 임의의 처리시간을 삽입할 수 있어 처리시간을 필요로 하는 요청에 대한 가상 실험을 통해 구현된 시스템의 성능평가도 가능하다. 실험결과 문서 크기 변화와 처리시간 변화는 구현된 시스템에서 서로 다른 영향을 끼치고 있으며, 이는 시스템의 성능평가에 중요한 요인이 됨을 알 수 있다.

또한 실험에 사용된 HTTP-WebServer는 포트를 통해 자료의 송수신이 이루어지고 주고받는 메시지 또한 변화가 없으므로 수정 없이 즉시 실험에 적용될 수 있다는 장점도 있다. 물론 포트를 이용한 통신의 경우 효율성에 문제가 있을 것으로 예상되므로 서비스관리자와 HTTP-WebServer간의 통신방식에 대한 평가가 필요하다.

3. 응용서비스 플랫폼 구현

3.1 시스템 장비

본 연구의 시스템은 LAN 환경에서 3개의 사설IP를 부여한 노드(Slave)들을 공유하고 실IP를 가진 하나의 Master 노드가 있다. 모든 노드는 100Mbps NIC와 노드간에는 dummy Hub로 연결되었으며, Master 노드를 제외한 노드(Slave)들은 모두 같은 사양의 노드이다. Master 노드의 경우는 CPU P4 1.8GHz, RDRAM 512Mbyte, HDD 14 Gbyte이고, Slave 노드의 경우는 CPU P2 200MHz, SDRAM 64Mbyte, HDD 2.4 Gbyte이다. 운영체제는 모두 Linux 9.x이고 메시지전달 라이브러리로 PVM 3.4.4를 사용하고 있다. Slave 노드들은 관리의 편리성을 위해 LC-KVM 104를 사용하였다.

3.2 시스템 구현

그림 3은 구현된 시스템의 구성을 나타낸다. 본 논문에서는 전위응용서버와 후위서버간의 통신은 PVM의 메시지 전달함수를 사용하였으며, 서버내부에서의 통신 중 서비스관리자와 서비스간의 통신은 서비스의 수정을 피하기 위해 TCP 통신을 이용하였다. 본 연구에서는 응용서비스로서 웹서버 기능을 간단히 구현한 공개 통신용 응용프로그램, 즉 특정 포트를 통해 통신을 하는 통신용 응용서버인 HTTP-

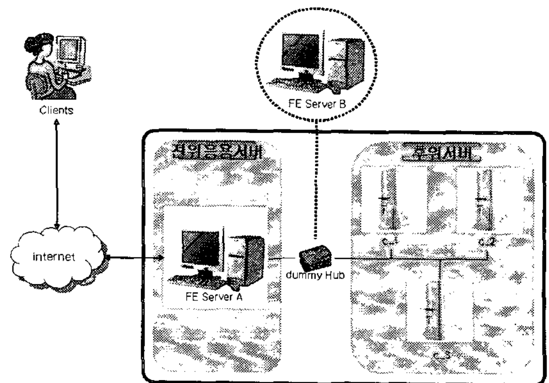

그림 3. 예제 시스템 구성

WebServer를 수정 없이 사용하였다. 본 연구를 위해서 사용된 HTTP-WebServer와 서비스관리자간의 통신방식의 비효율성을 개선하기 위해서는 PVM기반의 메시지 전달방식과 성능비교를 통해 통신비용을 개선하는 작업이 필요하다.

전위응용서버 A는 특정 응용서비스를 제공하고, 클라이언트의 요청을 처리, 분배하는 서비스관리자와 부하정보 유지 및 부하변화에 따른 시스템 재구성을 담당하는 부하관리자, 그리고 응용서비스인 HTTP-WebServer를 가진다. 클러스터된 후위서버군(c_1, c_2, c_3)은 작업관리자와 전위응용서버가 제공하는 응용서비스와 동일한 응용서비스를 가지고 있으면서 전위응용서버의 고부하시 해당 응용서비스를 제공하게 된다. 전위응용서버 A의 부하관리자는 부하정보를 바탕으로 후위서버의 PVM 데몬을 사용하여 해당 후위서버에 필요한 응용서비스를 생성하거나 중지시킨다. 현재 본 연구는 하나의 전위응용서버에 대해 실험을 했으며, 향후에는 또 다른 전위응용서버인 Master2를 추가하여 클러스터 자원을 동적으로 사용할 수 있는 관리기법에 대한 연구를 수행할 계획이다.

구현을 위해 고려한 시스템의 동작은 크게 네 개의 유형으로 나누어질 수 있다. 첫 번째는 사용자의 요청을 전위응용서버에서 처리할 경우로 기존의 단일서버 환경과 유사하다. 두 번째는 전위응용서버의 부하가 증가하여 클러스터된 후위서버군중 하나의 후위서버 사용가능 자원으로 등록하고 이를 이용하여 요청을 처리하는 경우이다. 세 번째 경우는 후위서버를 이용하는 중에 지속적으로 부하가 증가하여 또 다른 후위서버를 추가로 사용하는 경우이다. 마지막으로 네 번째는 부하가 감소하여 하나의 노드를 제거하는 경우이다. 즉 하나 이상의 노드를 이용하여 처리하는 중에 부하가 낮아져서 전위응용서버가 가지고 있는 후위서버중 하나를 다시 유휴자원으로 반환하는 경우이다. 각 경우에 따른 처리절차는 다음과 같다.

(1) 전위응용서버에서 처리할 경우

부하가 낮을 경우는 단일서버 시스템과 유사한데, 우선 클라이언트로부터 요청이 들어오면(①) 서비스관리자는 이 요청을 받아서(②) 새로운 쓰레드를 생성한다.(③) 이 쓰레드는 소켓을 생성하여 웹서비스에게 요청을 전달하고(④), 웹서비스는 이 요청을 받아(⑤) 적절한 처리를 한 후, 다시 역순으로 반환한다. 여기에서 모든 통신은 TCP/IP기반의 소켓 통신을 이용하고 있다. 그림 5는 전위 응용서버에서 요청을 처리할 경우, 서비스관리자의 pseudo code이다.

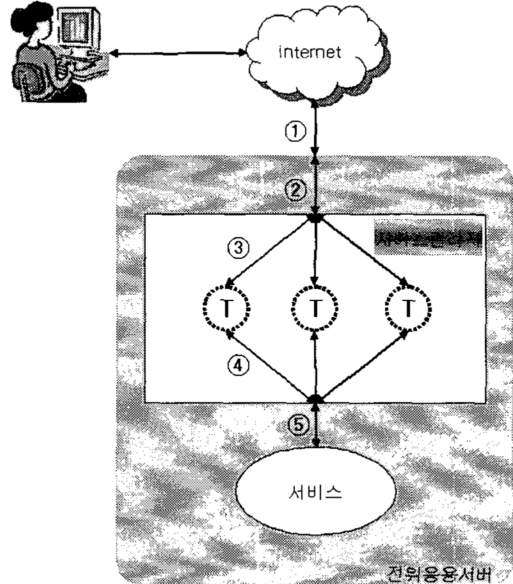

그림 4. 전위응용서버에서 요청 처리

(2) 전위응용서버의 부하가 지속적으로 증가하는 경우

전위응용서버는 고부하 상태가 되면 후위서버군의 자원을 사용해야 하는데, 이때 전위응용서버의 고부하 상태를 인식하는 프로세스가 부하관리자이다. 부하관리자는 주기적으로 전위응용서버의 부하 상태를 체크하여 고부하로 판정되면(①), 자신의 back_end_lists 중 적절한 노드를 선택하여 해당 후위서버의 작업처리자를 통해 필요한 서비스를 작동시킨다.(②) 그런 다음, 부하관리자는 전위응용서버의 서비스관리자에게 선택된 노드를 child_list에 추가하라는 메시지를 보내 서비스관리자의 child_list의 정보를 갱신한다.(③) 이 child_list에 노드 정보가 삽입되면 이후의 요청은 child_list에 해당하는 노드로 보내어진다. ④-⑥ 과정은 클라이언트로부터 요청을 받아들이는 과정이고, ⑦-⑨ 전위응용서버에서 선택된 후위서버의 서비스에게 요청을 전달하는 과정이다. 물론 처리결과는 ④-⑨의 역 과정을 통해 해당 클라이언트에게 반환된다. 그림 7은 후위서버 추가시 부하관리자와 서비스관리자의 pseudo code이다.

```
{ ...
 //클라이언트의 연결을 수락
 clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_addr, & clnt_addr_size);
 //쓰레드를 생성하여 clnt_connection으로 분기
 pthread_create(&thread, NULL, clnt_connection, &clnt_sock);

      { ...
        //로컬의 웹서비스와 연결
        connect(sock, (struct sockaddr*)&service_addr, sizeof(service_addr));
        //클라이언트의 요청을 받아서 웹서비스에게 전달
        while((str_len = read(clnt_sock, buf, BUFSIZE)) != 0)
          write(sock, buf, str_len);
        //웹서비스의 결과를 받아서 해당 클라이언트에게 전달
        while((str_len = read(sock, message, BUFSIZE)) != 0)
          write(clnt_sock, message, str_len);
      ... }

... }
```

그림 5. 전위응용서버에서 처리될 경우의 pseudo code

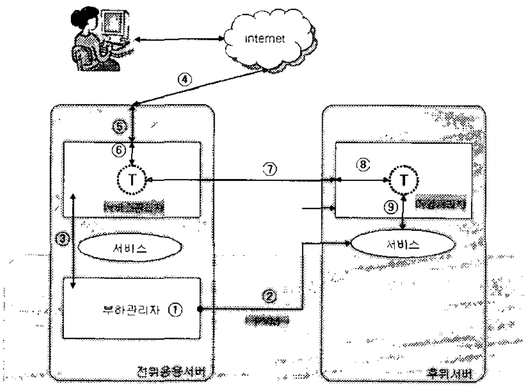

그림 6. 후위서버의 노드 추가하는 동작순서

(3) back_end_list에 등록된 후위서버의 부하가 증가하는 경우

세 번째 경우는 두 번째의 노드 추가 방법과 같으며, 단지 전위응용서버의 child_list에서 이전의 list들 이후에 새로운 노드가 추가된다. 새로운 노드가 사용 가능한 자원으로 추가되면 그 이후의 요청들은 이전 노드와 추가된 노드에게 나뉘어 분배된다.

(4) back_end_list에 등록된 후위서버의 부하가 감소하는 경우

부하가 낮아져서 현재 노드 하나를 삭제하는 경우이다. 이러한 경우의 핵심은 요청을 처리하는 도중에는 그 요청을 모두 처리하는 동안 기다렸다가 PVM을 이용하여 생성한 해당 서비스를 중지시켜야 한다는 것이다. 즉, 전위응용서버로부터의 요청 전달을 먼저 중지한 후 삭제할 노드가 더 이상 요청을 처리하지 않을 때 완전히 유휴한 상태로 환원시킨다. 이를 위해, 본 시스템은 다음과 같이 구성하였다.

그림 8은 하나의 노드를 삭제하는 방법을 나타내고 있다. 먼저 부하관리자가 부하가 낮으므로 하나의 노드를 삭제해도 된다고 판단하면(①), 전위응용서버의 서비스관리자에게 등록된 후위서버중 삭제시킬 노드 정보를 전달한다(②). 그러면 전위응용서버의 서비스관리자는 삭제 대상이 되는 후위서버의 작업처리자에게 더 이상의 요청을 전달하지 않는다(③). 그리고, 부하관리자는 삭제 대상의 후위서버의 작업처리자에게 메시지를 보내어 더 이상 처리할 요청이 없으면 메시지를 보내라고 지시한다(④). 작업처리자는 만약 처리 중인 요청이 있으면 정상적인 처리절차를 수행한 후 결과를 전위응용서버의 서비스관리자에게 넘겨주고(⑤,⑥), 모든 작업이 끝나면 전위응용서버의 부하관리자에게 모든 작업이 끝났다고 알려준다(⑦). 그러면, 전위응용서버의 부하관리자는 PVM을 통해 해당 노드의 제거하고(⑧), 최

```
{ ...
 //전위 응용 서버의 부하가 증가하면(부하분산 정책에 따라) 후위서버들 중 하나를 추가
 pvm_addhosts(&hosts[num_child], 1, &infos)
 //PVM을 통해 추가된 후위서버의 작업처리자와 웹서비스를 작동(스폰)시킴
 if(pvm_spawn("httpd", (char**)0, PvmTaskHost, hosts[num_child], 1, &tid[num_child][0]) ==1)
 if(pvm_spawn("smd", (char**)0, PvmTaskHost, hosts[num_child], 1, &tid[num_child][1]) ==1)
 //전위 응용 서버의 서비스 관리자에게 추가된 후위서버에 대한 정보를 보냄
 smd_connection("add", hosts[num_child]);
... } → 전위 응용 서버의 부하관리자 main() func.

{ ...
 //부하관리자로부터 메시지를 받아 child_list[]를 갱신
 if(strcmp(opr, "add") == 0)
 child_list[child_list_num++] = addr;
 else if(strcmp(opr, "del") == 0)
 child_list[--child_list_num] = '
 else printf("Invalid message");
... } → 전위 응용 서버의 서비스관리자 load_connection() func.
 //child_list[]가 갱신되면 서비스관리자는 child_list[]에 해당하는 후위서버에게 클라이언트의 요청을 전달
 //요청을 받은 후위서버는 그림 4와 같이 작동하고, 요청결과는 다시 서비스관리자에게 반환
```

그림 7. 후위서버 추가시 pseudo code

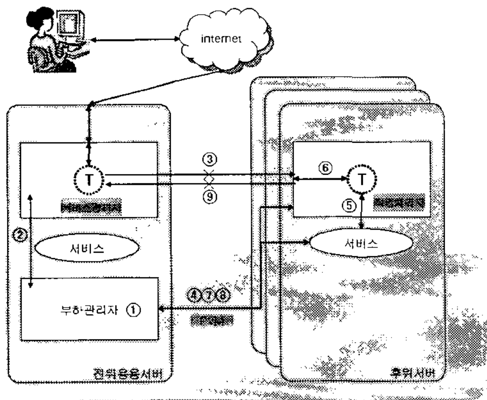

그림 8. 후위서버 노드 삭제 과정

종적으로 전위응용서버와 후위서버의 관련성은 더 이상 유지되지 않는다. (⑨). 물론 이 단계에서 후위서버의 서비스만을 중지시키는 것이 현실적이다. 그림 9는 후위서버 삭제 시 부하관리자의 pseudo code이다.

3.3 부하에 따른 부하분산 정책

부하분산 정책은 시스템의 성능에 주요한 요인으로 작용하므로 시스템에 적합한 부하분산 정책이 필요하다. 본 논문에서 제안한 시스템은 동적으로 시스템을 재구성하므로 어느 시기에 가용자원을 추가 및 삭제를 하는 것이 효율적인가의 문제는 시스템의 성능에 결정적인 영향을 끼칠 수 있다. 단일 기준으로 고부하와 저부하를 구분하는 경우에는 부하의 변화에 따른 유휴자원의 추가 및 삭제가 빈번하게 발생하므로 이러한 과정에 시스템의 오버헤드로 작용할 가능성이 높다. 따라서 본 논문에서는 그림 10과 같이 부하에 따라 두 개의 기준을 설정하여 유휴자원의 추가삭제가 빈번하게 발생하는 것을 방지하는 부하분산 정책을 제시, 채택하였다.

먼저, 부하에 따라 High, Medium, Low로 구분하고 Low Load에 해당하는 사용중인 후위서버가 있을 경우, 해당 후위서버의 부하가 Medium이 될 때까지 요청을 계속 전달한다. 해당 후위서버의 부하가 Medium Load 수준으로 올라가면 요청은 등록된 리스트상의 다음 후위서버에게 전달된다. 후위서버의 부하가 모두 Medium Load 수준이면 Round-Robin 방식으로 요청을 전달한다. High Load 수준의 후위서버들은 요청을 전달받지 않는데, 등록된 모든 후위서버가 High Load가 되면 그 때, 부하관리자가 후위서버 추가 단계를 거치게 된다. 또한, 모든 후위서버가 Low Load일 경우는 유휴한 후위서버를 삭제한다.

이렇게 함으로써, 한 개의 후위서버가 overflow될

```
{ ...
 //전위 응용 서버에 추가된(child_list) 후위서버의 부하를 측정(고부하)
 //마지막에 추가된 후위서버의 삭제 메시지를 전위 응용 서버의 서비스관리자에게 보냄
 //child_list에서 삭제된 후위서버는 더 이상의 요청을 받지 않게 됨
 smd_("del", hosts[num_child]);
 //해당 후위서버에게는 child_list에서 삭제되었다고 알려줌
 //이 메시지를 받은 후위서버는 자신이 처리하고 있는 요청을 모두 처리하면 전위 응용 서버의 부하관리자에게
   더 이상의 요청이 없다고 알려줌
 //후위서버로부터 모든 요청을 처리했다는 메시지를 받은 부하관리자는 PVM을 통해 해당 후위서버의 프로세스
   를 정지시킴.
 pvm_kill(tid[num_child][0]); //해당 노드의 웹서비스 정지
 pvm_kill(tid[num_child][1]); //해당 노드의 작업처리자 정지
 pvm_delhosts(&hosts[num_child],1, &infos);
... } → 전위응용서버의 부하관리자 main() func.
```

그림 9. 후위서버 삭제 pseudo code

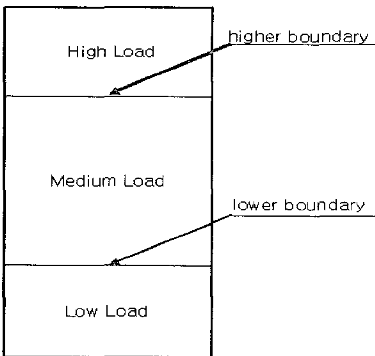

그림 10. 부하에 따른 부하 분산 정책

때마다 후위서버가 추가되는 것을 막고, 특정 후위서버의 부하가 많거나 적을 경우 해당 후위서버의 부하의 수준을 다른 후위서버와 맞추기 위해 요청을 전달하거나 전달하지 않게 된다.

이것의 문제점은 부하의 기준을 어디에 두느냐에 따라 달라지는데, 본 시스템에서 추구하는 다양한 서비스를 지원하는 플랫폼의 경우 서로 다른 요청에 따라 부하의 정도가 서로 다를 수 있으므로 많은 실험 연구를 통한 통계 값을 참조해야 한다. 또한, Low, Medium, High의 경계에 대한 근거도 실험 연구를 통해 가장 적절한 값을 구해야 한다. 현재 가장 효율적인 값을 찾기 위해 실험 연구를 계속하고 있다.

4. 실험 및 고찰

4.1 실험 환경

본 논문에서 구현한 인터넷 서비스 시스템은 실제 환경에서 테스트하여야하지만 이럴 경우 본 시스템과 관계없이 네트워크 상황에 따라 측정된 통계치가 달라질 수 있다. 그래서 본 논문에서는 시스템만의 정확한 통계치를 얻기 위해 외부와 연결되지 않은 가상IP를 가진 컴퓨터를 클라이언트로 활용하고, 이를 통해 가상적인 요청을 생성하여 서버에 전달하도록 하여 실험하였다. 이를 위해서 웹서버의 성능평가 도구인 httperf[14,15]를 이용하였다. httperf는 서버와 포트, 요청 파일, 초당 요청 수, 전체 연결 수, timeout 등의 값이 선택 가능하며, 그것에 대한 결과값을 볼 수 있다. 이를 이용해, 본 논문에서 구현한 응용서비스 시스템의 성능을 평가하였다.

4.2 실험 및 고찰

실험은 동적으로 구성되는 환경에 대한 실험을 위해 다음과 같은 실험 시나리오를 구성하여 구현된 시스템의 성능평가를 시도하였다. 우선 전위응용서버에서 서비스를 처리하는 경우와 후위서버가 하나씩 추가될 경우의 성능 차이를 실험하였으며, 이후 본 논문에서 구현한 시스템의 구성 요소간의 오버헤드를 측정하여 이전 실험결과에 대한 결과 평가작업을 수행하고, 실험결과를 바탕으로 구현된 시스템 가지는 장단점을 분석하여 적용 가능한 응용서비스의 종류에 대한 결론은 도출하였다.

실험은 서로 다른 옵션을 준 세 가지 실험으로 구성하였다. 실험 1은 요청된 파일의 크기가 고정된 상태에서 요청 수의 증가에 따른 응답속도를 측정하였다. 실험 2는 요청율이 고정된 상태에서 요청된 파일

크기에 따른 응답속도를 측정하였다. 마지막 실험 3은 앞선 실험 1, 2의 결과를 바탕으로 CPU의 처리시간을 추가로 요구하는 요청의 증가에 따른 응답속도를 측정하였다. 각 실험에 대한 검토 및 고찰은 다음과 같다.

실험 1: Request 수 증가에 따른 실험

우선 요청 수를 증가시키면서 본 논문에서 구현한 클러스터기반 응용서버의 응답속도를 측정하였다. 이를 위해 응용서비스만으로 구성된 단일서버 시스템과 응용서버의 동작 모드 중 전위응용서버만 존재하는 경우, 전위응용서버가 하나의 후위서버를 사용하는 경우, 전위응용서버가 두 개의 후위서버를 사용하는 경우, 그리고 전위응용서버가 3개의 후위서버를 사용하는 경우, 즉 후위서버의 추가에 따른 응답속도의 변화를 고찰하였다.

표 1과 그림 11은 공통적인 조건으로 요청파일을 1Kbyte로 고정하였고, 직접 응용서비스 서버에 요청했을 경우와 구현된 응용서비스 시스템에서 후위서버 사용 상태(후위서버의 수 = 0(Master), 1(1 Slave), 2(2 Slave), 3(3 Slave))에 따른 응답속도를 나타낸다. 실험은 요청율을 100에서 300까지 증가시키면서 수행되었다. 그 결과 구현된 시스템의 경우 후위서버를 추가하여 사용한 경우에 따른 이득은 미미하며, 단일서버의 경우에 비해 성능이 더 악화되었음을 표 1을 통해 알 수 있다. 이것은 하나의 요청을 처리하는 시간이 너무 짧고, 요청율 또한 단일서버에 오버헤더를 일으킬 만큼 충분하지 않아 관리자들의 동작에 따른 오버헤드가 부하의 분산에 따른 이득을 상쇄함을 알 수 있다. 즉 요청 처리시간이 요청 전달시간에 비해 짧은 응용의 경우 구현한 시스템이 적용되기 어려움을 보여주고 있다. 구현된 응용서버 시스템의 성능 개선을 위한 연구가 필요함을 확인하였다.

표 1. Request 수 증가에 따른 결과

Request (요청수)	Only Master	구현 프로세스 적용			
		Master	1' Slave	2' Slave	3' Slave
100	0.8	1.2	5.5	5.5	5.4
150	0.7	1.1	5.4	5.6	5.4
200	0.7	1.1	5.5	5.6	5.3
250	0.7	1.2	5.3	5.3	5.5
300	0.7	1.2	5.6	5.4	5.5

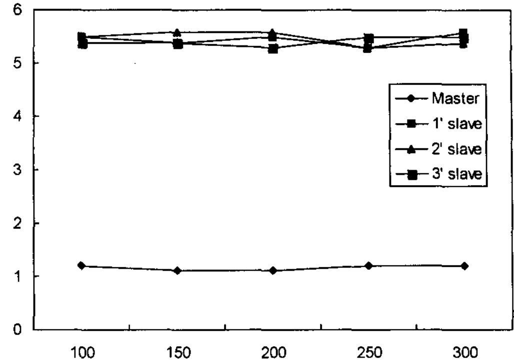

그림 11. Request 수 증가에 따른 응답속도

실험 2: File Size 변화에 따른 실험

본 실험은 초당 요청율을 '1'로 고정하고 전체 요청수가 100일 때, 요청 파일의 크기를 증가시킬 경우(10Kbyte - 500Kbyte) 응답속도를 측정한 것이다. 그 결과, 본 논문에서 구현한 프로세스를 적용한 실험에서 요청 파일의 크기가 400K를 초과하게 되면 오버헤드가 발생해 시스템의 응답속도가 갑자기 증가했다. 그리고, 약간의 차이지만 후위서버의 수가 많을수록 응답속도가 약간 빠름을 알 수 있다. 이 실험의 결과는 표 2와 그림 12와 같다. 실험결과에서 보이는 바와 같이 전송되는 자료의 양이 증가하면 결국 구현된 응용서버의 관리자간의 통신비용의 증가가 전체 시스템의 성능을 저하시키는 요인으로 작용함을 보여주고 있다. 따라서 내부 클러스터간의 통신을 개선하기 위한 연구가 필요함을 보이고 있다.

실험 1, 2의 결과를 통해 요청율의 증가나 파일크기 변화의 경우는 현재의 클러스터기반 서버에서는 오버헤드가 부하분산으로 인한 이득을 상쇄함을 알 수 있다. 이러한 원인으로 앞선 두 실험은 CPU 처리시간은 극히 미미하고 네트워크 트래픽 시간이

표 2. File Size 변화에 따른 결과 (단위: Kbyte)

File Size	Only Master	구현 프로세스 적용			
		Master	1' Slave	2' Slave	3' Slave
10	0.7	2.0	9.5	9.9	8.1
100	0.8	1.8	15.9	16.2	16.4
200	0.7	2.7	15.7	17.3	17.5
300	0.7	2.9	15.0	16.1	15.4
400	0.7	3.3	30.6	27.3	34.2
500	0.7	2.4	655.5	549.1	488.1

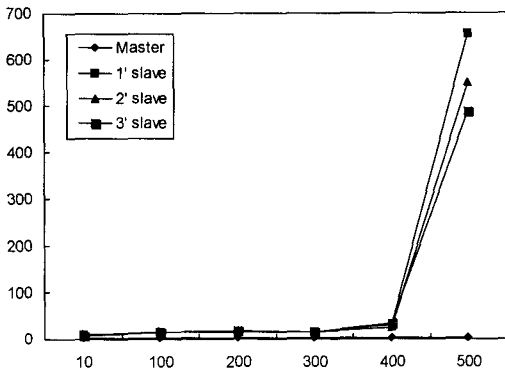

그림 12. File Size 변화에 따른 응답속도

응답속도의 거의 대부분을 차지하여 전체 응답시간을 증가시키는 요인으로 작용함을 알 수 있었다. 이에 대한 보다 구체적인 분석을 위해서 각 요청마다 요청을 처리할 때의 처리시간을 설정하고 후위서버를 사용하지 않는 경우(Master)와 3개의 후위서버를 사용하는 경우(3' Slave) 일 때의 응답속도에 대한 실험을 수행하였다.

실험 3: CPU 처리시간을 요구하는 Request에 대한 실험

실험 3을 위해, 실험결과에 대한 정확한 고찰을 위해 우선적으로 구현한 프로세스가 가지는 오버헤드를 측정해보았다.

그림 13의 왼쪽 그림은 일반적인 단일서버에서 Httpd 서비스를 제공하는 경우 1Kbyte 파일을 요청하는 경우의 오버헤드를 측정한 것이다. 오른쪽 그림은 본 논문에서 구현한 프로세스를 적용했을 경우의 오버헤드를 나타내고 있다. 그림 13에서 왼쪽의 Master와 오른쪽의 Slave의 httpd의 처리속도 차이는 3.1절에서 실험 장비 설명에서 Master와 Slave 후위서버의 성능 차이로 인해 발생하는 차이이다. 여기에서 우리는 단일서버에서 작동하는 경우와 구현된 시스템간의 성능이 약 3배 정도의 차이가 생기는 것을 볼 수 있다. 즉 구현된 시스템은 처리시간이 짧은 응용의 경우이거나 전송되는 문서의 양이 많은 경우 오버헤드가 증가하게 되어 전체적인 시스템의 성능이 저하시키는 요인으로 작용함을 확인할 수 있었다.

이러한 결론은 처리시간이 증가할 경우에 대한 실험이 필요함을 보이고 있다. 따라서 본 논문에서는 한 요청 당 10ms의 작업처리가 요구되는 경우에 대한 실험을 수행하였다. 실험 1과 2의 경우에는 각 Request 마다 네트워크 사용량에 비해 CPU 처리시간이 너무 짧아서 클러스터 자원을 사용하는 이점을 얻을 수 없었으므로 실험 3에서는 각 요청 당 10ms의 처리시간이 사용되는 경우 구현된 시스템의 요청 증가에 따른 성능을 평가하였다.

아래의 표 3과 그림 14는 10ms의 CPU 처리시간을 추가한 경우의 실험결과이다.

표 3과 그림 14는 각 Request들이 크기가 1K인 파일을 요청할 때, 초당 요청율을 '1'에서 '250'까지 증가시키면서 응답속도를 기록한 것이다. 본 실험 결과, 요청율이 '80' 미만 일 경우에는 전위응용서버만으로 처리하는 경우(Master)가 3개의 후위서버를 사용하는 경우(3' Slave)보다 응답속도가 더 빠르다는 것을 보이고 있는데, 이는 후위서버와 전위응용서버

표 3. CPU의 처리시간을 추가한 결과

Rate	Master	3' Slave
1	11.0	16.7
50	12.2	15.6
80	48.7	15.9
100	100.8	16.1
150	163.5	16.7
200	204.2	81.6
250	292.3	117.2

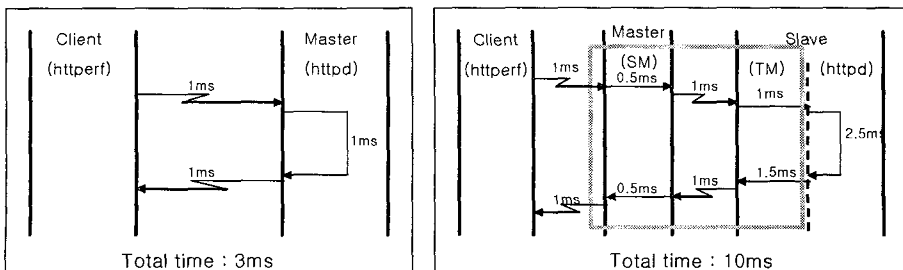

그림 13. 프로세스의 오버헤드 측정

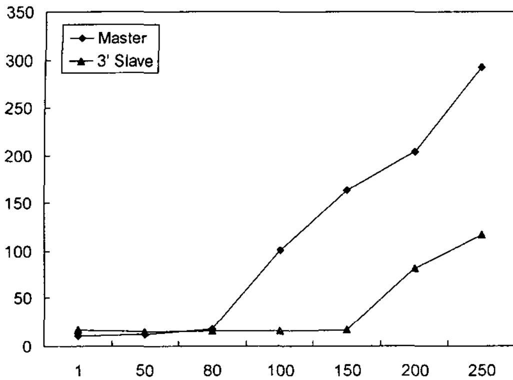

그림 14. CPU의 처리시간을 추가한 응답속도

와의 성능차이와 전위응용서버에서 후위서버로 요청을 전달하는 과정에서 수반되는 오버헤드에 기인한 것으로 판단된다. 요청율이 '80'이상이 되면 3개의 후위서버를 사용하는 경우(3' Slave)가 안정적인 응답속도를 보임을 알 수 있다. 이것은 CPU의 처리시간이 필요한 작업을 하는 경우 하나의 CPU(Master)로 처리하는 것 보다 3개의 CPU(3' Slave)로 분산시켜 처리함으로써 응답속도 개선의 이득을 얻을 수 있음을 보이고 있다. 즉 이러한 결과를 통해서 요청의 특성, 즉 처리시간, 통신비용이 응용서비스의 성능에 결정적인 영향을 끼침을 알 수 있으며, 본 논문에서 구현한 응용서비스 시스템은 처리시간을 필요로 하는 응용의 경우에 적합함을 알 수 있다. 또한 기존의 응용서비스 시스템과의 성능차이를 극복하기 위해서는 구현된 응용서비스 시스템의 오버헤드를 감소시킬 수 있는 연구가 필요함을 보여준다.

5. 결론 및 향후과제

본 시스템은 부하의 증감에 관계없이 모든 클라이언트들에게 안정된 서비스를 제공해주는 응용서비스 시스템이다. 이것은 부하에 무관하게 균일한 양질의 서비스를 제공하는 것으로, 이를 위해 공유 클러스터 자원을 사용하고 있으며, 이 자원을 PVM을 통해 동적으로 사용해서 한 응용서비스에 제한하지 않고 다양한 응용서비스를 제공할 수 있다.

본 시스템은 네트워크 트래픽을 많이 요구하는 요청보다 CPU의 처리시간이 높은 요청의 처리에 적절하다. 즉, 실험 1,2의 결과에서 보이는 바와 같이 CPU 처리시간이 짧고 통신비용이 높은 작업에서는 얻어지는 이득이 미미하지만 실험 3의 결과를 통해 알 수 있는 바와 같이 CPU의 처리시간이 증가할 경우에는 본 시스템을 통한 이득이 기대됨을 알 수 있었다. 또한 본 시스템은 기존의 웹서비스를 지원하기 위해서는 구현된 시스템의 통신비용을 줄일 수 있는 방안과 구현된 시스템의 효율성을 제고할 수 있는 연구가 추가로 수행되어야 함을 보이고 있다. 이를 위해서 향후에는 TCP/IP기반의 쓰레드 통신이 아닌 모든 통신을 메시지 전달 라이브러리인 PVM을 이용하는 시스템의 구현 및 재평가 작업이 필요하며, 더불어서 PVM의 요소기능에 대한 성능평가 및 성능개선 작업 또한 필요할 것을 판단된다.

그리고 현재 구현된 시스템의 경우는 PVM의 Master에 응용서비스가 탑재되어 하나의 응용서비스를 제공하고 있으나 향후에는 다양한 서비스를 하나의 공유 클러스터에서 지원할 수 있는 클러스터 관리기법에 대한 연구도 필요하다고 판단된다. 그리고 본 시스템이 실제 응용에 적용되기 위해서는 부하분산 정책도 중요한 성능개선 요인이므로 이에 대한 실험적 연구가 필요하다고 수행되어야 할 것이다.

참 고 문 헌

[1] Ian Foster and Carl Kessleman, "The Grid : Blueprint for a New Computing Infrastructure," Morgan Kaufmann Publishers. Inc., 1999.

[2] Rajkumar Buyya, "High Performance Cluster Computing : Architecture and Systems, Volume 1," Prentice Hall, 1999.

[3] 윤찬현 외 4인, "인공심장의 혈류해석을 위한 Globus 기반 협업 기술 개발," ITRC Forum 2002, Seoul, Korea, May. 2002.

[4] W. Gropp, "Learning from the success of MPI," High Performance Computing - HiPC 2001, number 2228 in Lecture Notes in Computer Science, pp 81-92, Dec. 2001.

[5] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM : Parallel Virtual Machine A User's Guide and Tutorial for Network Parallel Computing," MIT Press, Cambridge, MA, 1994.

[6] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM : Parallel Virtual Machine: A Users Guide and Tutorial for Networked Parallel Computing. Scientific and Engineering Computation," MIT Press, Cambridge, MA, USA, 1994.

[7] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High Performance, Portable Implementation of the MPI Message Passing Interface Standard," Parallel Computing, Volume 22, number 6, pp 789-828, Sep. 1996.

[8] The C10K Problem, "http://www.kegel.com/c10k.html"

[9] Om P. Damani, P. Emerald Chung, Yennun Huang, "ONE-IP: Techniques for Hosting a Service on a Cluster of Machines," In Proc. the Sixth International WWW Conference, http://decweb.ethz.ch/WWW6/Technical/Paper196/Paper196.html, Apr. 1997.

[10] "http://www.pdl.cmu.edu/Pasis/survivablestorageindex.html"

[11] 권세오, 김상식, "리눅스 클러스터형 웹 서버 설계," 한국정보과학회지 제18권 3호, pp. 48-56, Mar. 2000.

[12] Greg Regnier, "CSP: A System-Level Architecture for Scalable Communication Services," Intel Technology Journal Q2, http://developer.intel.com/technology/itj/q22001.htm, May, 2001.

[13] 서대화, 민병준, 이기욱, "네트워크 연결형 스토리지의 기술 동향," 한국정보과학회지, 제19권 3호, pp 6-13, Mar. 2001.

[14] The httperf Tool, "ftp://ftp.hpl.hp.com/pub/httperf/"

[15] David Mosberger and Tai Jin, "httperf - A Tool for Measuring Web Server Performance," In Proc. the SIGMETRICS Workshop on Internet Server Performance, pp 59-67, Jun. 1998.

김 동 근

2002년 2월 인제대학교 컴퓨터공학부(학사)
2004년 2월 인제대학교 컴퓨터공학부(석사)

관심분야 : 분산처리, 유비쿼터스컴퓨팅, 웹서비스

박 세 명

1994년 8월 경북대학교 전자공학과, 전산공학 전공(Ph.D)
1990년 3월 ~ 현재 인제대학교 컴퓨터공학부 교수

관심분야 : 분산처리, 유비쿼터스컴퓨팅, Grid컴퓨팅, 지능형홈시스템