인터넷 환경에서 서버간 부하 분산을 위한 새로운 웹 클러스터 기법

김 승 영† · 이 승 호††

요 약

본 논문에서는 능동적으로 서버의 상태 파악이 가능하고 서버의 운영체제에 의존적이지 않은 디스패처 방식을 개선하여 인터넷 환경에서 서버간 부하 분산을 위한 새로운 웹 클러스터 기법을 제안한다. 제안한 새로운 웹 클러스터 기법은 자율적 부하 분산 기능과 트랜잭션 페일 세이프(Transaction Fail-safe) 기능을 갖는다. 자율적 부하 분산 기능은 기존의 균등 분배와 고정 분배 부하 분산 방식을 개선하여 웹 클러스터 기법이 동작되는 상황에서 서버 각각의 부하 정도에 따라 실시간으로 트래픽(Traffic) 분배율을 조정하는 기능이다. 트랜잭션(Transaction) 페일 세이프 기능은 트랜잭션 단위의 복구가 지원되지 않는 기존의 방식을 개선하여 연속된 일련의 트랜잭션이 처리되는 도중 발생된 서버측 장애를 복구하여 주는 기능이다. 본 논문에서 제안한 새로운 웹 클러스터 기법은 유닉스 운영체제 환경에서 C 언어로 구현하였고, 기존의 상용 웹 클러스터 솔루션과의 벤치마크 비교를 통해 성능을 비교 분석하였다. 브로드케스팅 방식과의 성능 비교에서는 트래픽 처리량이 많아질수록 제안한 새로운 웹 클러스터 기법의 성능이 우수하였다. 라운드 로빈 DNS 방식과의 성능 비교에서 트래픽 처리 성능은 비슷하였으나, 서버의 장애 상황에서는 제안한 새로운 웹 클러스터 기법이 트래픽을 보다 신뢰적으로 처리 하였다. 따라서 본 논문에서 제안한 새로운 웹 클러스터 기법을 인터넷 서비스에 적용할 경우 급격히 증가하는 서비스 요청과 이로 인한 서버의 과부하를 효율적으로 처리하여 보다 신뢰적인 서비스가 가능할 것으로 기대된다.

A New Web Cluster Scheme for Load Balancing among Internet Servers

Seung-young Kim† · Seung-Ho Lee††

ABSTRACT

This paper presents a new web cluster scheme based on dispatcher which does not depend on operating system for server and can examine server's status interactively. Two principal functions are proposed for new web cluster technique. The one is self-controlled load distribution and the other is transaction fail-safe. Self-controlled load distribution function checks response time and status of servers periodically, then it decides where the traffic goes to guarantee rapid response for every query. Transaction fail-safe function can recover lost queries including broken transaction immediately from server errors. Proposed new web cluster scheme is implemented by C language on Unix operating system and compared with legacy web cluster products. On the comparison with broadcast based web cluster, proposed new web cluster results higher performance as more traffic comes. And on the comparison with a round-robin DNS based web cluster, it results similar performance at the case of traffic processing. But when the situation of one server crashed, proposed web cluster processed traffics more reliably without lost queries. So, new web cluster scheme proposed on this dissertation can give alternative plan about highly increasing traffics and server load due to heavy traffics to build more reliable and utilized services.

키워드 : Web Cluster, 자율적 부하 분산 기능, 트랜잭션 페일 세이프 기능, 디스패처(Dispatcher) 방식

1. 서 론

인터넷 사용자의 폭발적인 증가로 인하여 월드 와이드 웹은 백본(Backbone) 트래픽을 매 분기당 2배로 증가해야 할 정도이다. 이에 따라 인터넷 서비스 제공업자(ISP : Internet Service Provider)들은 새로운 회선 비용으로 매일 수백만 불을 지출하고 있는 실정이다. 그럼에도 새로운 사용자와 멀티미디어 어플리케이션(Multimedia Application)의 성장을 따라 잡지 못하고 있다. 이제까지의 인터넷 성장은 이러한 측면에서 백본의 대역폭(Band-width) 증가에 초점을 맞추어 왔으나, 근래에 들어서는 서버(Server)의 수용 한계를 넘어선 트래픽의 증가로 인해 서버의 성능(Performance)과 가용성(Availability)을 높이는 방안이 더욱 중요시되고 있다.

시스템의 성능을 향상시키는 방안으로서 과거에는 높은 성능의 프로세서(Processor)를 장착하거나 슈퍼컴퓨터와 같이 여러 개의 CPU(Central Process Unit)를 채택한 병렬 처리(Parallel Processing) 기법[1]을 사용하였다. 그러나, 단일

† 준 회 원 : 오픈버드 eSolution Biz Unit 선임연구원
†† 정 회 원 : 국립 한밭대학교 전기 · 전자 · 제어공학부 교수
논문접수 : 2001년 6월 28일, 심사완료 : 2001년 12월 20일

서버로의 성능향상은 물리적인 확장의 한계로 인하여 네트워크(Network) 트래픽의 증가추세를 따라오지 못하고, 성능대비 투자비용이 기하급수적으로 증가되어서 효율성이 매우 떨어졌다.

근래에는 병렬 처리 기법의 한계를 뛰어넘어 시스템의 성능을 향상시키기 위한 방법으로 네트워크에 접속된 여러 개의 서버를 하나로 연결하는 클러스터(Cluster)[2] 기법이 대두되고 있으며 월드 와이드 웹 서비스 기반하의 클러스터 기법을 웹 클러스터(Web Cluster)라 한다. 웹 클러스터가 갖추어야 할 기능으로는 다수의 서버에 연산을 분배 공유할 수 있는 부하 분산 기능[3], 특정 서버의 문제가 전체 서비스에 영향을 미치지 않도록 하는 페일 세이프(Fail Safe)기능[4], 특정 서버의 갱신 내용을 다른 서버에 전달하기 위한 실시간 동기화 기능[5] 등이 있다. 한편, 웹 클러스터를 구현하는 방식으로는 구현 아키텍처(Architecture)에 따라 라운드 로빈 DNS(Round-robin DNS), 브로드케스팅(Broadcasting), 직접 라우팅(Direct Routing), 디스패처(Dispatcher) 방식 등으로 나뉘어진다.

라운드 로빈 DNS 방식[6]은 사용자가 해당 서비스의 도메인 주소(Domain Address)를 DNS 서버를 통해 IP 주소로 변환하는 과정에서 변환을 요청할 때마다 여러 서버의 다른 IP를 알려 주어 부하를 분산하는 방식이다. 장점으로는 서버의 운영체제에 비종속적이며 적은 기회 비용으로 손쉽게 구축이 가능하다. 단점으로는 구현 특성상 매번 연결을 위해 서버의 IP를 질의하기 때문에 이에 따른 연결 지연(Delay)이 존재하고, DNS 서버가 웹 클러스터 서버의 장애여부를 파악하지 못하므로 서버 장애에 따른 페일 세이프 기능이 구현되지 못한다.

브로드케스팅[7] 방식은 물리적으로 네트워크에 도달한 트래픽을 모든 서버에 전달시키고 그 중 특정한 한 서버만이 응답을 하는 방식으로, 이더넷(Ethernet)의 브로드캐스팅(Broadcasting) 특성을 이용한 방식이다. 장점으로는 병목현상을 발생시키는 단일 지점이 없게 된다. 단점으로는 서버 운영체제의 이더넷 어뎁터(Adapter)[29]를 수정하여야 하기 때문에 GNU/Linux와 같이 소스코드가 공개되고 수정이 허용되는 운영체제이외에는 적용될 수가 없고, 네트워크상의 모든 패킷이 모든 서버에서 수신되어야 하기에 불필요한 트래픽이 증가하게 된다.

직접 라우팅[8] 방식은 요청을 중계해 주는 서버가 존재하여 클라이언트의 요청을 서비스 서버에 연결하여 주는 방식이다. 클라이언트의 최초 요청은 중계 서버를 거쳐 서버에 연결이 되지만 요청에 대한 서버의 응답은 클라이언트와 직접 통신하는 특성을 갖는다. 장점으로는 중계 서버의 트래픽 집중화를 막을 수가 있으며, 중계 서버의 다양한 트래픽 분배 정책에 따라 트래픽을 어느 정도 효율적으로 분배할 수 있게 된다. 단점으로는 브로드케스팅 방식과 같은 이유로 서버의 운영체제에 제약이 있으며, 요청에 대한 응답은 직접 클라이언트와 이루어지기 때문에 서버의 트래픽 처리량을 정확하게 계산할 수가 없게 된다.

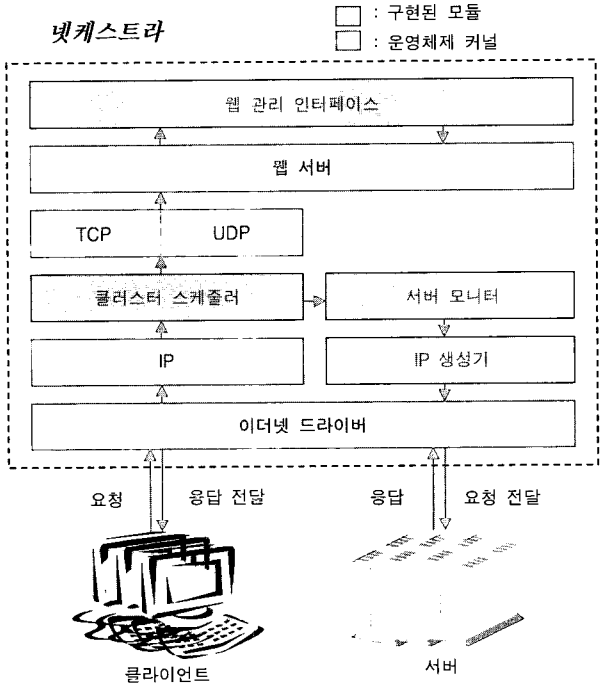
디스패처 방식[9]은 직접 라우팅 방식과 구조적으로 유사하게 웹 클러스터 앞단에 중계 서버가 존재한다. 그러나 직접 라우팅 방식과는 다르게 모든 클라이언트의 요청과 서버의 응답은 중계 서버를 통한다는 특성을 갖는다. 장점으로는 모든 요청과 응답을 중계 서버가 감시하기 때문에 서버의 상태 파악이 수월하고, 서버의 이더넷 어뎁터를 수정할 필요가 없으므로 운영체제에 종속적이지 않고 자유롭다. 단점으로는 중계 서버가 모든 트래픽을 처리하여야 하므로 단일 병목 지점으로 작용할 수가 있게 된다.

본 논문에서는 능동적으로 서버의 상태 파악이 가능하고 서버의 운영체제에 의존적이지 않은 디스패처 방식을 개선한 인터넷 환경에서 서버간 부하 분산을 위한 새로운 웹 클러스터 기법을 제안한다. 제안한 새로운 웹 클러스터 기법은 서버의 응답 시간 및 상태 등을 능동적으로 파악하여 서버의 성능에 따라 트래픽을 분배해주는 자율적 부하 분산 기능[10]과 트랜잭션 처리 도중 발생된 서버측 장애에 대처하여 트랜잭션을 복구하여 주는 트랜잭션 페일 세이프 기능[11] 등을 갖는다. 자율적 부하 분산 기능은 웹 클러스터 기법이 동작되는 상황에서 서버 각각의 부하 정도에 따라 실시간으로 트래픽 분배율을 조정하는 기능이다. 기존의 부하 분산 방식은 서버의 부하를 측정치 않고 트래픽을 균등 분배하거나 일정한 비율로 고정 분배를 하므로, 트래픽의 성격에 따라 특정 서버에 부하가 집중될 수 있는 단점이 있다. 그러나 본 논문에서 제안한 자율적 부하 분산 기능은 서버의 부하 정도에 따라 실시간으로 트래픽 분배량을 조정하여 각각의 서버가 최상의 성능을 발휘할 수 있도록 유도하며, 전체 트래픽을 빠르게 처리하도록 해준다. 트랜잭션 페일 세이프 기능은 연속된 일련의 작업이 처리되는 도중 발생된 서버측 장애를 복구하여 주는 기능이다. 기존의 상용 웹 클러스터 기법들은 부하 분산 기능에만 초점을 맞추어 페일 세이프 기능이 제공되지 않거나, 각각의 연결에 대한 페일 세이프 기능만이 제공되기 때문에 연속된 트랜잭션의 처리 도중에 장애가 발생될 경우 트랜잭션은 버려지게 된다. 따라서 금융권과 같이 트랜잭션의 처리가 보장되어야 하는 경우엔 사용될 수가 없었다. 반면에 본 논문에서 제안한 트랜잭션 페일 세이프 기능은 서버에서 처리중인 트랜잭션의 처리 과정이 감시되기 때문에 서버측 장애가 검출되면 해당 트랜잭션을 다른 서버로 위임하여 처리한다. 웹 클러스터는 단일 서버에 비해 장애가 일어날 확률이 높기 때문에 제안된 트랜잭션 페일 세이프 기능은 웹 클러스터의 신뢰를 더욱 향상시키게 된다.

2. 본 논문에서 제안하는 새로운 웹 클러스터 기법

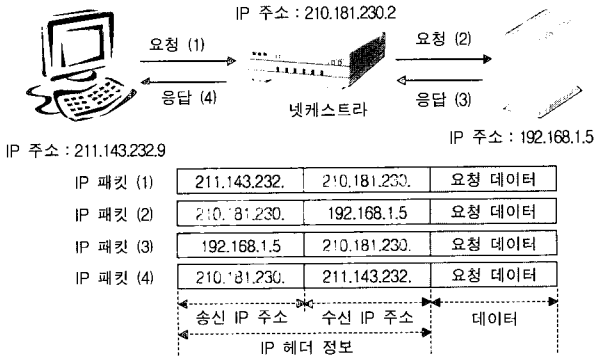
본 논문에서 제안하는 새로운 웹 클러스터 기법인 넷케스트라는 (그림 1)과 같이 웹 클러스터 스케줄러, 서버 모

니터, IP 생성기, 웹 관리 인터페이스로 구성된다. 넷케스트라는 IP 패킷 단위의 처리를 하며, 클라이언트와 서버의 모든 요청과 응답은 넷케스트라를 경유하게 된다. 클라이언트에게 실제 웹 클러스터 서버 자체는 감추어지게 되며, 클라이언트의 입장에서는 넷케스트라를 서버로 인식하여 모든 클라이언트는 넷케스트라의 단일 IP 주소만으로 통신을 취하게 된다.



(그림 1) 넷케스트라의 전체 구성도

한편, 클라이언트에서 송신된 IP 패킷이 처리되는 일련의 과정은 다음과 같다. 클라이언트의 요청 IP 패킷은 먼저 운영체제의 이더넷 드라이버가 수신을 하게 된다. 이 단계의 패킷은 실제 IP 프레임(IP Frame)이라고 명명되는 이더넷 레벨의 데이터 조각이기 때문에 이더넷 드라이버는 IP 프레임의 헤더를 제거한 후 IP 레이어로 전달하고, 웹 클러스터 스케줄러까지 도착하게 된다. 웹 클러스터 스케줄러는 서버 모니터의 정보를 참조하여 적절한 서버에 IP 패킷을 배정한 후 IP 생성기에게 IP 패킷을 전달한다. 이때, 추후 서버의 응답을 다시 클라이언트에 전달해 주기 위해 클라이언트 IP 주소와 배정된 서버의 IP 주소 그리고 IP 패킷의 일련번호 정보를 임시로 보관해 둔다. IP 생성기에 도착하는 IP 패킷의 헤더엔 (그림 2)의 IP 패킷 (1)과 같이 클라이언트 IP 주소가 송신 주소로 입력되어 있고, 수신 주소엔 넷케스트라의 IP 주소가 입력되어 있으므로, 이를 수정하여 송신 주소엔 넷케스트라의 IP 주소를 저장하고 수신 주소엔 배정된 서버의 주소를 저장하여 (그림 2)의 IP 패킷 (2)와 같이 수정한 후 이더넷 드라이버를 통해 해당 서버로 송신한다. 이때 IP 패킷의 정보가 변경되었기 때문에 새로운 정보에 맞는 CRC Check Sum을 다시 계산하여 저장하는 역할도 IP 생성기에서 같이 처리한다. 서버에 전달된 IP 패킷 조각들이 의미 있는 정보로 재조합되면 서버는 요청에 대한 처리를 시작하고 결과에 대한 응답 IP 패킷을 넷케스트라에게 다시 송신하게 된다. 이때 IP 패킷의 헤더 정보엔 (그림 2)의 IP 패킷 (3)과 같이 송신 주소에 서버 IP 주소를 수신 주소에 넷케스트라 주소가 입력되어 있다. 서버에서 응답한 IP 패킷은 넷케스트라의 이더넷 드라이버를 거쳐 IP 레이어 그리고 웹 클러스터 스케줄러에 도착하게 된다. 웹 클러스터 스케줄러는 송신자 주소가 등록된 서버의 IP 주소일 경우엔 이를 응답 IP 패킷으로 간주하고 저장해 두었던 정보를 참고하여 요청을 보낸 클라이언트의 IP 주소와 함께 해당 IP 패킷을 IP 생성기로 전달한다. IP 생성기는 수신된 IP 패킷의 헤더 정보를 (그림 2)의 IP 패킷 (4)와 같이 송신자 주소에 넷케스트라의 IP 주소를, 수신자 주소에 클라이언트의 IP 주소로 설정하고 클라이언트에게 전달한다.



(그림 2) 넷케스트라를 경유하는 IP 패킷의 상태별 헤더 정보

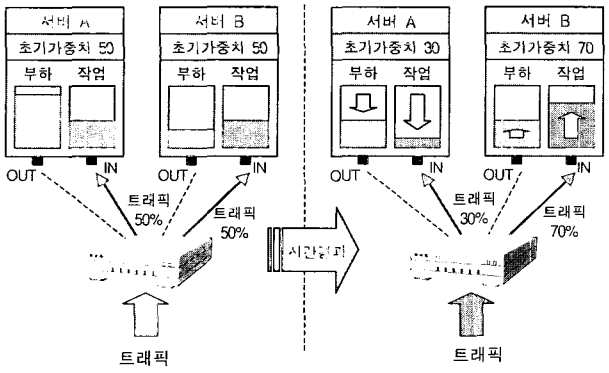
2.1 자율적 부하 분산 기능

자율적 부하 분산 기능은 넷케스트라가 동작하는 상황에서 서버 각각의 부하 정도에 따라 실시간으로 트래픽 분배율을 조정하는 기능으로, 트래픽의 집중화를 방지하고 효율적으로 처리될 수 있도록 한다. 제안하는 자율적 부하 분산 기능을 구현하기 위하여 넷케스트라는 내부적으로 가중치 테이블을 관리한다. 가중치 테이블은 서버의 응답 시간을 기초로 실시간 계산되며, 가중치 테이블에 기초하여 트래픽을 서버에 분배한다.

(그림 3)는 가중치 테이블에 의한 트래픽 분배 방식을 보여준다. 초기에 넷케스트라가 구동된 직후엔 서버의 응답시간과 수용 능력에 대한 통계 자료가 수집되기 전이므로 관리자가 임의로 설정한 초기 가중치 값에 의거하여 트래픽을 분배한다. 예를 들어 (그림 3)의 좌측 그림은 2대의 웹 클러스터 서버를 구동하였을 경우인데, 서버 A와 서버 B에 초기 가중치를 각각 50%씩 설정하였다. 넷케스트라가 구동된 직후엔 설정된 가중치에 따라 전체 트래픽의 50%는 서

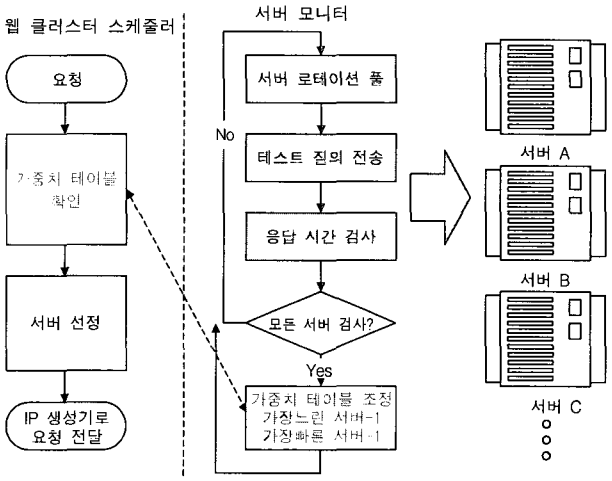
버 A에, 나머지 50%는 서버 B로 분배하였는데, 서버 A에 분배된 트래픽이 서버 B에 분배된 트래픽보다 높은 부하를 발생하여서 서버 A에 부하가 집중되는 현상이 발생되고 있다. 시간이 경과함에 따라 넷케스트라는 서버의 부하 정도에 따라 초기 가중치 값을 (그림 4)의 흐름도에 따라 조정을 하게 되는데 (그림 3)의 우측 그림이 이에 해당한다. 서버 A의 부하가 높음을 감지한 넷케스트라는 서버 A의 가중치 값을 30% 까지 낮추고, 서버 B의 가중치 값을 70% 까지 높여서 서버 A와 서버 B의 부하가 같은 수준을 유지하도록 하고 있음을 보여준다.

이처럼 넷케스트라는 트래픽이 요청될 때마다 가중치 테이블을 참고하여 서버를 배정하게 되는데, 넷케스트라는 이를 위해 서버 중 가장 적은 가중치를 1로 하여 트래픽 분배율을 계산한다. 4와의 우측과 같은 경우 서버 A의 가중치는 30%이고 서버 B의 가중치는 70% 이므로, 설정된 트래픽 분배율은 가중치가 가장 적은 서버 A가 1, 서버 B는 2(소수점 반올림)로 분배율이 계산된다. 넷케스트라는 계산된 트래픽 분배율(1 : 2)에 따라 입력된 트래픽을 서버 A, B, B, A, BB …와 같이 배정한다. 만약 서버의 가중치가 25% : 75%까지 증감하게 되면 트래픽 분배율은 다시 1 : 3으로 조정이 되며 입력된 트래픽은 A, B, B, B, A, B, B, B … 순으로 배정하게 된다.



(그림 3) 자율적 부하 분산 기능 동작 예제

이때 넷케스트라의 가중치를 조정하는 방식은 (그림 4)와 같다. 넷케스트라는 주기적으로 서버의 장애 유무와 요청에 대한 응답 시간을 점검한 후 응답 시간이 가장 느린 서버의 가중치를 1 감소시키고, 가장 빠른 서버의 가중치를 1 증가시키는 작업을 반복한다. 시간이 흐름에 따라 가중치 테이블은 서버 A의 응답속도와 서버 B의 응답속도가 거의 같게 되도록 조정이 된다. 이것은 일반적인 웹 클러스터의 트래픽 분배 방식과 다른데, 일반적인 웹 클러스터는 서버의 부하와 상관없이 트래픽을 균등 분배하거나 일정한 비율로 고정 분배를 하므로, 특정 서버에 부하가 집중될 수 있는 위험이 있다. 특정 서버에 부하가 집중되면 해당 서버는 요청에 대한 느린 응답을 보이게 되고 결국 전체 서버를 효율적으로 활용하지 못하게 된다. 반면에 넷케스트라는 서버의 부하가 일정하게 유지되도록 트래픽 분배율을 실시간 적으로 조정하기 때문에 요청에 대한 응답 시간이 보장되고 전체 서버의 성능을 효율적으로 활용할 수 있게 된다.

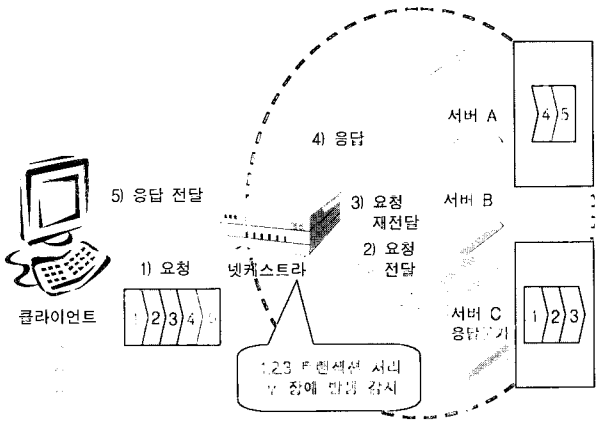


(그림 4) 자율적 부하 분산 기능 동작 예

2.2 트랜잭션 페일 세이프 기능

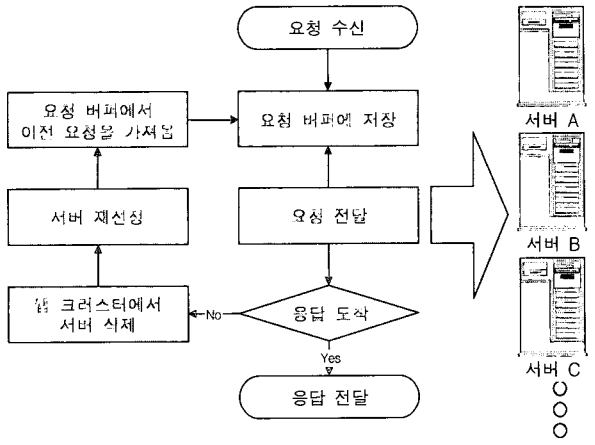
트랜잭션 페일 세이프 기능은 연속된 일련의 작업이 처리되는 도중 발생된 서버측 장애를 복구하여 주는 기능이다. 트랜잭션이란 연속된 작업의 묶음으로 모두 처리되지 않거나 반드시 모두 처리되어야 한다. 또한 페일 세이프 기능은 웹 클러스터 서버 중 특정 서버에 장애가 발생되었을 때 해당 서버의 장애 요인이 제거될 때까지 트래픽을 전달하지 않는 기능이다. 일반적인 클라이언트의 연속된 작업 요청은 여러 대의 서버에 분산하여 처리하여도 좋지만, 트랜잭션 요청은 작업의 연속성을 위하여 초기에 배정된 서버로만 전달을 해 줄 필요가 있다. 본 논문에서는 이것을 웹 클러스터의 트랜잭션 처리 기능이라고 하고, 트랜잭션 처리 중에 발생된 서버의 장애에서도 트랜잭션의 처리를 보장해 주는 기능을 트랜잭션 페일 세이프라 정의한다. 예를 들어 (그림 5)은 5개의 연속된 IP 패킷을 갖는 트랜잭션을 처리하고 있다. 작업 (1)이 웹 클러스터에 요청되었을 때 서버 C가 배정되었고, 연속된 트랜잭션의 작업들은 계속 서버 C로 위임되어 처리되고 있다. 이때 (4)번째 작업을 처리하던 중 서버 C에 장애가 발생하였는데, 넷케스트라는 해당 트랜잭션에 대한 작업 서버를 서버 A로 다시 배정하고 (4) 번 작업을 서버 A에 재 요청하여 트랜잭션 처리를 완료하고 있다. (그림 5)은 지금까지의 설명을 도식화하여 보여준다.

한편, 서버의 장애 여부는 서버의 응답시간으로 측정하며, 이때 기준이 되는 응답 시간은 일반적인 응답 시간보다 매우 크게 설정된다. 넷케스트라의 트랜잭션 페일 세이프 기능은 (그림 6)과 같은 흐름으로 구현된다. 넷케스트라는 서버



(그림 5) 트랜잭션 페일 세이프 기능 동작 예

의 급작스러운 장애로부터 트랜잭션을 잃어버리지 않기 위해 서버의 응답이 확인될 때까지 요청을 보관하고 있다가, 응답이 확인되면 클라이언트에 응답을 전달하고 보관된 요청을 파기한다. 그러나 서버의 장애 상황으로 인하여 지정된 응답 시간 안에 응답이 수신되지 않으면 다른 서버로 보관된 요청을 재 전달하여 트랜잭션의 처리를 계속한다.



(그림 6) 트랜잭션 페일 세이프 기능 흐름도

3. 실험 및 고찰

본 논문에서 제안한 인터넷 환경에서 서버간 부하 분산을 위한 새로운 웹 클러스터 기법인 넷케스트라의 성능을 객관적으로 측정하고 비교 평가하기 위하여 <표 1>과 같이 1대의 디스패처와 3대의 서버를 독립된 100Mbps 패스트 이더넷(Fast Ethernet) 세그먼트(segment)에 구성하였다. 또한 넷케스트라를 FreeBSD(Free Berkeley System Distribution) 유닉스 운영체제에서 C 언어로 구현하였으며, 성능 측정을 위해 사용한 벤치마크(Benchmark)로는 WebBench 3.0[12]을 사용하였다. WebBench 3.0은 해당 서버의 웹 서비스 능력을 측정하는 벤치마크로써 저명한 월간지 Byte를 비롯 국내외로 가장 많이 사용되는 웹 서버 성능 측정 도구이다.

〈표 1〉 실험에 사용된 장비

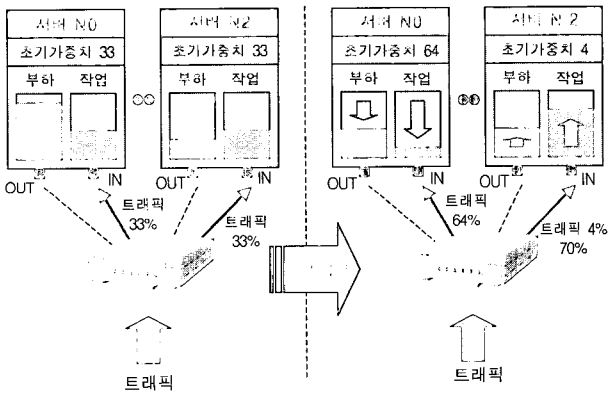
장치명	연산장치	메모리	디스크
넷케스트라	펜티엄 III 500MHz	512MB	18GB SCSI
서버 A	펜티엄 III 500MHz	256MB	18GB SCSI
서버 B	펜티엄 III 500MHz	256MB	18GB SCSI
서버 C	펜티엄 III 500MHz	256MB	18GB SCSI

3.1 넷케스트라 성능 실험

3.1.1 자율적 부하 분산 기능 실험

넷케스트라의 자율적 부하 분산 기능을 검증하기 위하여 3대의 서버에 <표 2>의 코드를 적용한 후 실험하였다. <표 2>는 동일한 요청에 대해서 임의로 각 서버의 부하 정도를 다르게 하기 위함인데, 이로 인해 동일한 요청에 대해서 서버 N0가 가장 부하가 적게 되고 서버 N2가 가장 많은 부하를 갖게 된다.

<표 3>은 넷케스트라의 자율적 부하 분산 기능에 대한 실험 결과이다. <표 3>의 결과를 살펴보면 넷케스트라를 구동한 직후엔 서버 N0, N1, N2가 모두 (그림 7)의 좌측과 같이 33%로 동일한 트래픽 분배율이 설정되었으나, 80초가 지난 후에는 (그림 7)의 우측과 같이 64%, 32%, 4%로 트래픽 분배 비율이 조정되어, 서버의 부하정도에 따른 적절한 트래픽 분배가 이루어짐을 알 수 있다. (그림 8)는 넷케스트라의 자율적 부하 분산 기능이 동작하는 과정을 넷케스트라에 연결된 터미널에서 관찰한 실험 화면이다.



(그림 7) 시간 경과에 따른 자율적 부하 분산 기능의 동작도

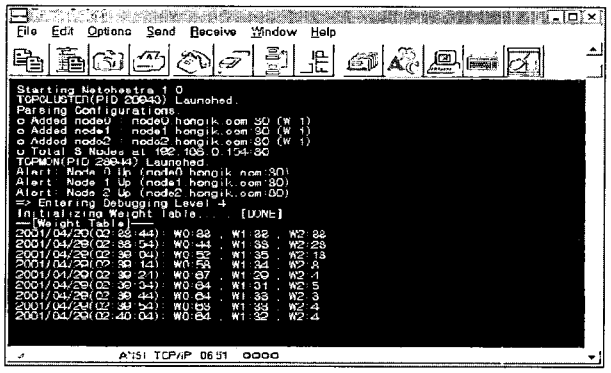
〈표 2〉 자율적 부하 분산 실험에 사용된 서버측 소스 코드

N0 서버에 적용된 코드	N1 서버에 적용된 코드	N2 서버에 적용된 코드

프로그램 시작 {
루프 100,000회
}
	프로그램 시작 {
	루프 10,000,000회
	}
		프로그램 시작 {
		루프 100,000,000회
		}

〈표 3〉 넷케스트라의 자율적 부하 분산 기능 실험

경과시간	서버 N0 가중치	서버 N1 가중치	서버 N2 가중치
0 초	33 %	33 %	33 %
10 초	44 %	33 %	23 %
20 초	52 %	35 %	13 %
30 초	58 %	34 %	8 %
40 초	67 %	29 %	4 %
50 초	64 %	31 %	5 %
60 초	64 %	33 %	3 %
70 초	63 %	33 %	4 %
80 초	64 %	32 %	4 %



(그림 8) 넷케스트라의 자율적 부하 분산 실험 화면

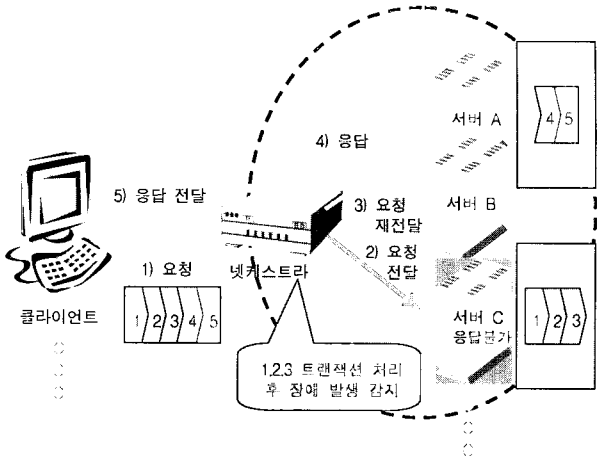
3.1.2 트랜잭션 페일 세이프 기능 실험

넷케스트라의 트랜잭션 페일 세이프 기능을 실험하기 위하여 넷케스트라를 운용하며 연결된 서버에 강제적으로 서비스 불능 상황을 만들고 페일 세이프 기능을 실험하였다. 실험에는 (그림 9)의 트랜잭션코드를 사용하였는데, (그림 5)의 트랜잭션은 총 5개의 연속된 작업으로 구성되어 있다.

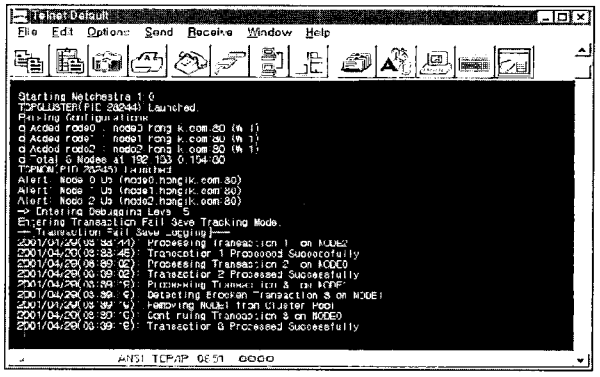
```
TestTransaction() {
    For ( SequenceNo = 1 ; SequenceNo <= 5 ; SequenceNo++) {
        tmpResult = SendRequest(Data[SequenceNo]) ;
        if(tmpResult < 0) Error(BROKEN_TRANSACTION) ;
        Result = Result + tmpResult ;
    }
}
```

(그림 9) 트랜잭션 페일 세이프 실험에 사용된 소스 코드

실험은 (그림 10)과 같이 4번째 작업이 진행되는 과정에서 강제로 서버의 장애상태를 만든 후 모든 트랜잭션의 정상처리 여부를 측정하였다. 넷케스트라는 트랜잭션 페일 세이프의 기능 동작을 추적하기 위해 (그림 11)과 같은 디버깅 정보를 출력하도록 구현되었고, 트랜잭션의 처리 과정은 해당 정보를 통해 확인할 수 있었다. (그림 11)은 3개의 트랜잭션에 대한 처리 과정을 나타내고 있는데, 1번째와 2번째 트랜잭션은 각각 서버 2(NODE0)와 서버 0(NODE0)에서 정상 처리되었다. 3번째 트랜잭션 처리는 서버 1(NODE1)이 배정되었는데, 트랜잭션 처리도중 서버 장애가 발생되었다. 넷케스트라는 서버 장애를 진단하였음을 알리는 메시지(Detecting Broken Transaction)를 출력하고 트랜잭션의 처리를 서버 0(NODE0)로 재 배정(Continuing Transaction 3 on NODE0)하여 트랜잭션 처리를 완료하였음을 알 수 있다.



(그림 10) 트랜잭션 페일 세이프 기능 동작 개념도



(그림 11) 트랜잭션 페일 세이프 실험 결과

3.2. 넷케스트라와 상용 웹 클러스터와의 성능 측정 결과

본 논문에서 제안한 넷케스트라와 다른 상용 웹 클러스터와의 성능측정은 객관성을 유지하기 위하여 동일한 실험 환경에서 수행하였다.

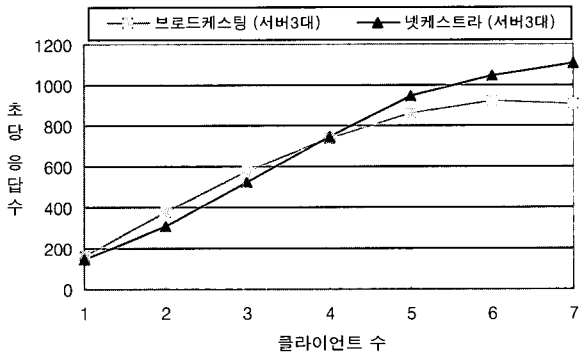
3.2.1 브로드케스팅 방식과의 성능 측정 결과

<표 4>는 본 논문에서 제안한 넷케스트라와 브로드케스팅 방식을 채택한 상용화된 웹 클러스터[13]와의 성능 측정 결과이고, (그림 12)는 성능 측정 결과를 그래프로 나타낸 것이다. <표 4>과 (그림 12)의 결과는 낮은 트래픽에서는 브로드케스팅 방식이, 높은 트래픽에서는 넷케스트라가 우수한 처리 성능을 나타내고 있다. 브로드케스팅 방식은 모든 트래픽에 대해 서버간의 통신과정을 거쳐 해당 트래픽에 대한 처리 서버를 자발적으로 결정하기 때문에, 적은 트래픽일 경우에는 효율이 좋으나 트래픽의 양이 증가하면 급격하게 증가하는 서버간 통신으로 인하여 처리 성능이 급격하게 저하된다. 반면에 넷케스트라는 트래픽이 디스패처를 경유하는 특성으로 인해 트래픽의 양이 적을 경우엔 이점이 약점으로 작용하고 있

으나, 트래픽의 양이 증가하였을 경우에도 브로드케스팅 방식에서와 같은 서버간의 트래픽 문제가 없기 때문에 결과적으로 높은 처리 성능을 보여주고 있다. 따라서 웹 클러스터에 연결된 서버의 개수가 많아지고 처리하는 트래픽의 양이 많을수록 넷케스트라의 성능이 우수할 것으로 사료된다.

〈표 4〉 브로드케스팅 방식과 넷케스트라와의 성능 측정 결과

클라이언트	브로드케스팅 (서버 3대)	넷케스트라 (서버 3대)
1 개	*159*	*142*
2 개	*382*	*308*
3 개	*581*	*520*
4 개	*739*	*754*
5 개	*862*	*957*
6 개	*920*	*1056*
7 개	*914*	*1113*



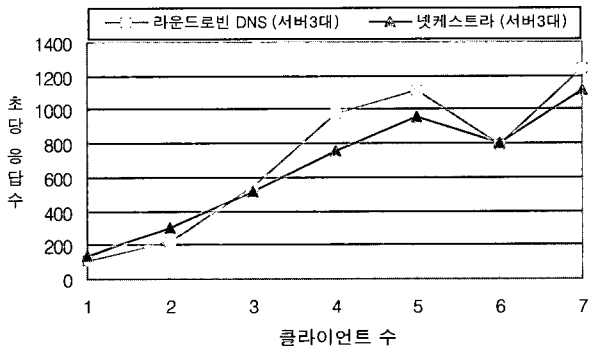
(그림 12) 브로드케스팅 방식과 넷케스트라와의 성능 측정 결과

3.2.2 라운드 로빈 DNS 방식과의 성능 측정 결과

<표 5>는 본 논문에서 제안한 넷케스트라와 라운드 로빈 DNS 방식을 채택한 상용화된 웹 클러스터[14]와의 성능 측정 결과이고, (그림 13)은 성능 측정 결과를 그래프로 나타낸 것이다. <표 5>와 (그림 13)의 결과는 낮은 트래픽에서는 넷케스트라가, 높은 트래픽에서는 라운드 로빈 DNS 방식이 우수한 처리 성능을 나타내고 있다. 라운드 로빈 DNS 방식은 클라이언트가 연결을 시도 할 때마다 네임서버에 질의를 하여야 하기 때문에, 적은 트래픽에 대한 처리 효율은 낮으나 트래픽의 양이 증가할 경우엔 네임서버의 캐시가 작용하여 효율이 높아진다. 반면에 넷케스트라는 트래픽이 적을 경우엔 라운드 로빈 DNS에서와 같은 지연 시간이 없기 때문에 효율이 좋으나, 트래픽의 양이 증가하면 디스패처를 경유하는 패킷의 지연시간 증가로 인하여 라운드 로빈 DNS 방식에 비해 상대적으로 낮은 성능을 보여준다. 한편, <표 5>와 (그림 13)에서 클라이언트의 개수가 6개째일 때의 결과는 한대의 서버에 임의로 장애 상황을 발생시킨 후, 트랜잭션 페일 세이프 기능이 동작하는 넷케스트라와 페일 세이프 기능이 없는 라운드 로빈 DNS의 성능 측정 결과이다. 성능 측정 표와 그래프 상의 초당 응답수치는 두 방식 모두 비슷하지만, 라운드 로빈 DNS는 페일 세이프 기능이 없으므로 요청의 약 1/3을 처리하지 않고 버리는 반면, 넷케스트라는 트랜잭션 페일 세이프 기능이 동작하여 나머지 2대의 서버에서 모든 요청을 처리하였다는 차이가 있다. 웹 클러스터는 다수의 서버로 구성이 되며, 이 때 서버의 장애가 발생할 확률은 단일 서버의 경우보다 현저히 높기 때문에 페일 세이프 기능이 없는 웹 클러스터는 성능과 안정성을 모두 만족시키지 못하므로 운용과 성장의 한계가 있다. 웹 클러스터에 있어 성능의 문제는 서버를 추가함으로 인해 보상 받을 수 있기 때문에 대규모 웹 클러스터에서는 웹 클러스터의 신뢰성이 더욱 민감한 문제이다. 넷케스트라와 라운드 로빈 DNS와의 성능 측정 결과에서 트래픽에 대한 처리 성능은 두 방식 모두 비슷한 성능을 보여주고 있으나, 본 논문에서 제안하는 새로운 웹 클러스터 기법인 넷케스트라는 서버의 장애 상황에서도 신뢰적으로 트래픽을 처리하고 있었다. 따라서, 넷케스트라가 라운드 로빈 DNS 방식에 비해 보다 우수하고 만족스러운 성능을 제공하리라 사료된다.

〈표 5〉 라운드 로빈 DNS 방식과 넷케스트라와의 성능 측정

클라이언트	라운드로빈 DNS (서버 3대)	넷케스트라 (서버 3대)
1 개	*109*	*142*
2 개	*223*	*308*
3 개	*541*	*520*
4 개	*973*	*754*
5 개	*1016*	*957*
6 개(서버 장애시)	*794*	*804*
6 개	*1202*	*1056*
7 개	*1247*	*1113*



(그림 13) 라운드 로빈 DNS 방식과 넷케스트라와의 성능 측정 결과

4. 결 론

인터넷 정보 제공자 및 사용자의 기하급수적 증가는 필연적으로 정보 제공 서버의 과부하를 초래하고 있다. 또한 웹을 포함한 인터넷 서비스가 전자 상거래와 같은 사업의 중요한 도구가 됨에 따라 인터넷 무정지 서비스는 매우 중요한 문제로 대두되었다. 이 문제에 대해 클러스터링 솔루션은 훌륭한 해결책이 될 수 있다.

본 논문에서는 능동적으로 서버의 상태 파악이 가능하고 서버의 운영체제에 의존적이지 않은 디스패처 방식을 개선한 인터넷 환경에서 서버간 부하 분산을 위한 새로운 웹 클러스터 기법인 넷케스트라를 제안하였다 넷케스트라는 기존의 상용 솔루션에 비해 다음과 같은 장점을 있다.

첫 번째로 넷케스트라의 자율적 부하 분산 기능은 기존 상용 솔루션들의 일반적인 트래픽 균등 분배와 고정 비율 분배에서 벗어나 서버의 성능에 따라 트래픽을 자율적으로 분배하여 전체 서버의 성능을 효율적으로 운용한다. 두 번째로 트랜잭션 페일 세이프 기능은 금융권과 같이 극단적으로 트랜잭션 처리가 중요한 경우에도 신뢰적인 서비스를 제공할 수 있도록 한다. 세 번째로 운영체제에 종속적이지 않기 때문에 이미 구축된 네트워크 환경을 손쉽게 웹 클러스터로 변이할 수 있다.

웹 클러스터는 시스템 오류에 대한 신속한 대처와 사이트의 트래픽 변화에 따른 유동적인 용량 확충이 근본이라는 점에서 기능과 성능 중 어느 하나도 소홀히 할 수 없는 부분이다. 이러한 면에서 본 논문의 넷케스트라는 기존의 상용 제품들보다 월등히 유연하고 신속하게 서버의 상태에 대처할 수 있고, 보다 신뢰적인 서비스의 구축이 가능하리라 사료된다.

넷케스트라와 같은 디스패처 방식의 단점은 디스패처가 다운되었을 때 서비스가 마비될 수 있다는 점이다. 따라서 앞으로의 연구 과제로는 주 디스패처가 다운되었을 때 전환 가능한 보조 디스패처를 구현하여 신뢰성을 더하고, 다수의 디스패처가 클라이언트의 요청을 분배 처리하는 병렬 디스패처 시스템에 대한 연구도 진행되어야 할 것으로 생각된다.

참 고 문 헌

[1] Morioka, M., Kurosawa, K., Miura, S., Nakamikawa, T., Ishikawa, S., "Design and evaluation of the high performance multi-processor server," Computer Design : VLSI in Computers and Processors, 1994. ICCD '94. Proceedings., IEEE International Conference, pp.66-69, 1994.

[2] Jian Liu, Longlu Xu, Baogen Gu, Jing Zhang, "A scalable, high performance Internet cluster server," High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition, Vol.2, pp.941-944, 2000.

[3] Cardellini, V., Colajanni, M., Yu, P. S., "Redirection algorithms for load sharing in distributed Web-server systems," Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference, pp.528-535, 1999.

[4] Chin Wen Cheong, Ramachandran, V., "Genetic based Web cluster dynamic load balancing in fuzzy environment," High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition, Vol.2, pp.714-719, 2000.

[5] Baldoni, R., Bonamoneta, S., Marchetti, C., "Implementing highly-available WWW servers based on passive object replication," Object-Oriented Real-Time Distributed Computing, 1999. (ISORC '99) Proceedings. 2nd IEEE International Symposium, pp.259-262, 1999.

[6] Kangasharju, J., Ross, K. W., "A replicated architecture for the Domain Name System," INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol.2, pp.660-669, 2000.

[7] Kangasharju, J., Ross, K. W., "A clustering structure for reliable multicasting," Computer Communications and Networks, 1999. Proceedings. Eight International Conference, pp.378-383, 1999.

[8] Dongeun Kim, Cheol Ho Park, Daeyeon Park, "Request rate adaptive dispatching architecture for scalable Internet server," Cluster Computing, 2000. Proceedings. IEEE International Conference, pp.289-296, 2000.

[9] Canal, R., Parcerisa, J. M., Gonzalez, A., "Dynamic cluster assignment mechanisms," High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium, pp.133-142, 1999.

[10] Kremien, O., Kramer, J., "Flexible load-sharing in configurable distributed systems," Configurable Distributed Systems, 1992., International Workshop, pp.224-236, 1992.

[11] Baker, W. E., Horst, R. W., Sonnier, D. P., Watson, W. J., "A flexible ServerNet-based fault-tolerant architecture," Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium, pp.2-11, 1995.

[12] ZDNet, "WebBench," http : //www.zdnet.com/ etestinglabs/ stories/benchmarks/0,8829,2326243,00.html.

[13] ㈜아라기술, "SmartIP Clustering Solution," http : //www.aratech. co.kr/.

[14] PolyServer, Inc., "Local Cluster," http : //www.polyserve.com/ products/localcluster21/.

김 승 영

e-mail : nobreak@openbird.com

1998년 홍익대학교 전자전산공학과 졸업(공학사)

2001년 국립 한밭대학교 대학원 전자공학과 졸업(공학석사)

2001년~현재 오픈비트 eSolution Biz Unit 선임연구원

관심분야 : 컴퓨터 통신, 인터넷, Embedded 통신 기기

이 승 호

e-mail : shlee@cad.hanbat.ac.kr

1986년 한양대학교 전자공학과 졸업(공학사)

1989년 한양대학교 대학원 전자공학과 졸업(공학석사)

1994년 한양대학교 대학원 전자공학과 졸업(공학박사)

1994년~현재 국립 한밭대학교 전기·전자·제어공학부 부교수

관심분야 : 컴퓨터 통신, 인터넷, 집적회로설계(CAD for VLSI), 시스템온칩설계