공학 석사 학위논문

리눅스 PC 클러스터를 이용한 웹-메일 서버 시스템에 관한 연구

The Research on Web-mail Server System
using a Cluster of Linux PCs

2001년 2월

서울대학교 대학원
전기 컴퓨터공학부

전 우 철

리눅스 PC 클러스터를 이용한 웹-메일 서버 시스템에 관한 연구

The Research on Web-mail Server System
using a Cluster of Linux PCs

지도교수 하 순 회

이 논문을 공학석사 학위논문으로 제출함.

2000년 10월

서울대학교 대학원
전기 컴퓨터공학부
전 우 철

전우철의 공학석사 학위논문을 인준함.

2000년 12월

위원장 전 주 식 인
부위원장 하 순 회 인
위원 장 래 혁 인

초록

이 논문은 리눅스기반의 개인용 컴퓨터 클러스터를 이용한 웹-메일 전자우편 서버 시스템에 관한 연구를 기술하고 있다.

'/etc/passwd'파일과 'procmail'을 이용한 기존 전자우편 서버시스템에서의 낮은 성능은 느린 MDA 프로그램과 동시에 연결 가능한 SMTP 커넥션 수의 제약 때문이라는 사실을 실험을 통해 확인할 수 있었다. 본 논문에서는 이러한 문제점을 해결하기 위해 MDA로부터 사용자의 우편함까지 전자우편이 배달되는 부분을 수정하였다. 성능 저하를 일으킬 수 있는 원인으로는 웹-메일 서비스를 제공하기 위해 사용한 데이터베이스와 잘못된 우편함 관리방식이 있었다.

결국 이러한 사실을 바탕으로 개발한 시스템은 기존 시스템에 비해 수신 성능을 약 두 배 가까이 향상시킬 수 있었다. 또한 리눅스 PC 클러스터로 웹-메일 시스템을 구현했으므로 동시에 연결 가능한 SMTP 커넥션 수가 증가한 것과 같은 효과를 볼 수 있는 장점이 있다.

주요어: 전자우편 서버 시스템, 클러스터, 우편함,
메일 딜리버리 에이전트, 리눅스 피씨

학번: 99419 - 542

목차

그림 목차

표 목차

1. 서론

전자우편은 인터넷에서 웹과 함께 중요한 위치를 차지하고 있는 서비스로 최근 인터넷이 급성장 하면서 그 사용자수와 사용량이 급증하고 있다. 전자우편 사용량이 증가하면서 서비스가 좋은 시스템에 사용자가 더욱 집중되는 경향이 있으므로[1], 다수의 소규모 서버 시스템보다는 소수의 대규모 서버시스템들이 나타나고 있는 추세이다. 실례로 전자우편 사용자 수는 미국 Earthlink사의 경우 전자우편 서비스 가입자수가 1996년 1월 25,000명에서 1997년 9월 350,000명으로 증가했고 곧 1,000,000명에 이를 것이라고 전망하고 있으며[2], Hotmail의 경우는 현재 75,000,000명에 이르고 있다.[1] 전자우편 사용량은 AOL, Hotmail과 같은 전세계적인 사이트들의 경우 이미 1998년 당시 하루에 약 10,000,000통 이상의 전자우편을 처리하고 있었으며[3], 이러한 증가추세는 앞으로 계속될 것으로 보인다.

지금까지는 이러한 대용량 서버 시스템을 구성하기 위해 가격이 매우 비싼 고성능 서버를 사용하는 것이 일반적이었다. 하지만 그 동안 PC의 가격은 낮아진 반면 성능은 높아져서 적은 수의 싼 PC들로 클러스터를 구성하더라도 원하는 성능의 시스템을 얻을 수 있게 되었고, network 속도도 10Mbps에서 100Mbps fast ethernet, Gigabit ethernet등으로 고속화되었으므로 현재는 이러한 문제점들이 없어진 상태이다.

그리고 이러한 PC클러스터 시스템을 사용할 경우 한 대의 고성능 서버를 사용하는 것보다 확장성과 가용성 측면에서 유리하다. 기존의 단일 서버 시스템에서 처리능력이 부족하면 새 서버를 구입해야 하고 그때까지 사용하던 서버는 더 이상 사용할 수 없었다. 반면에 클러스터 시스템은 원래의 시스템에 노드를 추가함으로써 처리능력을 확장할 수 있다. 또 기존의 단일 서버 시스템은 서버에 문제가 생기면 서비스를 제공할 수 없지만, 클러스터 시스템은 한 노드에 문제가 생기더라도 나머지 노드들이 동작하고 있으므로 성능의 저

1) 2000년 Hotmail 광고에서의 자체발표

하는 일어날 수 있어도 서비스 중단은 일어나지 않는다.

이와 같이 단일 서버 시스템에 비해 PC클러스터 시스템이 비용, 확장성, 가용성 측면에서 유리하고, 그 동안 PC클러스터 구성에 장애가 되어왔던 문제점들도 해결되었으므로 PC클러스터를 이용한 대용량 전자우편 시스템에 대한 요구가 증가하고 있다.

그러나 그러한 요구가 높음에도 불구하고 대용량 웹서버에 대한 연구에 비해 대용량 전자우편 서버에 대한 본격적인 연구는 별로 없었고[3], 웹서비스가 서버 시스템에서 정보를 읽어 가는 동작을 주로 하는데 비해 전자우편 서비스는 서버 시스템에 정보를 쓰는 동작이 상당한 부분을 차지하고 있으므로 전자우편 서버 시스템에 대한 연구가 필요함을 알 수 있다[4].

본 논문에서는 리눅스 PC클러스터를 이용한 대용량 웹-메일 서버 시스템의 프로토타입 개발에 대한 연구 결과를 제시하고 프로토타입과 기존 시스템의 성능 비교 결과를 보이고자 한다. 특히 MDA(Mail Delivery Agent)[2]로부터 사용자의 우편함(mailbox)에 전자우편이 배달되기까지의 과정에 초점을 두고 우편함(mailbox) 관리 부분과 MDA(Mail Delivery Agent)프로그램을 수정한 프로토타입을 구현하였고 웹-메일 서비스를 지원하기 위해 전자우편 서버에 웹-메일 관련 부분을 추가하고 웹-서버용 클라이언트를 제작하였다.

여기서 공개운영체제인 리눅스를 사용한 것은 비용이 적게 드는 클러스터의 장점을 살리기 위한 것이고 웹-메일을 주요 목표로 한 것은 최근 전자우편 사용자들이 전자우편 서비스를 이용하는 방식이 웹-메일이기 때문이다.

본 논문은 기존 전자우편 서버 시스템의 수신 성능 저하의 원인으로 알려진 MDA(Mail Delivery Agent)의 처리 시간과 동시에 연결 가능한 SMTP connection수의 제한에 대한 사실을 확인하고 이러한 사실을 바탕으로 수신 성능이 향상된 리눅스 PC 클러스터를 이용한 웹-메일 서버 시스템에 대한 연구 결과를 제시한다.

논문의 구성은 다음과 같다. 2장 관련연구에서는 크게 전자우편 서

2) 본 논문의 '2.1.1 전자우편의 전달과정'에서 설명하고 있다.

버와 클러스터라는 두 가지 범주로 구분하여 각각의 개략적인 소개와 기존의 연구들을 정리 분석한다. 3장 구현에서는 본 논문에서 구현한 프로토타입 시스템에 대한 자세한 설명을 한다. 4장 실험 결과에서는 기존 시스템의 수신 성능 저하 원인을 확인하는 실험과 시스템의 수신 성능 측정 실험에 대해서 그 결과와 분석을 보인 뒤, 5장에서 결론을 짓는다.

2. 관련 연구

본 논문에서 구현한 시스템은 일반적인 linux시스템의 전자우편 전달과정 중에서 MDA(Mail Delivery Agent)가 사용자의 우편함(mailbox)에 전자우편을 배달하는 과정에 초점을 두고 우편함과 MDA프로그램을 수정한 것이다. 여기서는 이러한 시스템의 이해에 필요한 배경 지식을 소개하고 기존의 연구들을 정리 분석함으로써 다음 장의 시스템 구현에 대한 이해를 돕는다. 먼저 논문에서 수정한 MDA와 전자우편 배달 과정을 중심으로 전자우편 시스템에 대한 개략적인 소개를 한다. 다음으로 대표적인 MTA(Mail Transfer Agent)인 'sendmail'프로그램에 대한 내용을 본다. 실제로 전자우편을 받아서 MDA에 전달하는 것은 MTA이므로 전자우편의 처리 과정은 MTA의 영향을 크게 받기 때문이다. 또 이러한 전자우편 시스템에 대한 지식을 바탕으로 전자우편 서버 시스템에 대한 기존의 연구를 정리 분석하여 각 접근 방식의 장단점을 파악한다. 마지막으로 여러 가지 클러스터 구성기법들을 봄으로써 그 장단점을 살펴본다.

2.1 전자우편 서버 시스템 관련 연구

전자우편 서버 시스템에 관한 기존의 연구들을 살펴보기 위해서는 전자우편 시스템에 대한 이해가 선행되어야 하며 특히 대다수의 서버에서 MTA(Mail Transfer Agent)로 사용하고 있는 'sendmail'[5] 프로그램에 대한 이해가 필요하다. 여기서는 실제로 송신자로부터 전자우편이 수신자에게 전달되는 과정의 예를 살펴보면서 전자우편 시스템에 대한 이해를 돕고 'sendmail'프로그램에 대해서 살펴본 다음, 기존의 연구들을 정리, 분석한다.

2.1.1 전자우편의 전달과정

송신자가 전자우편을 보내서 수신자가 전자우편을 읽기까지의 과

정은 그림1과 같고 이러한 전달과정은 크게 다음과 같은 다섯 가지 과정으로 나눌 수 있다.

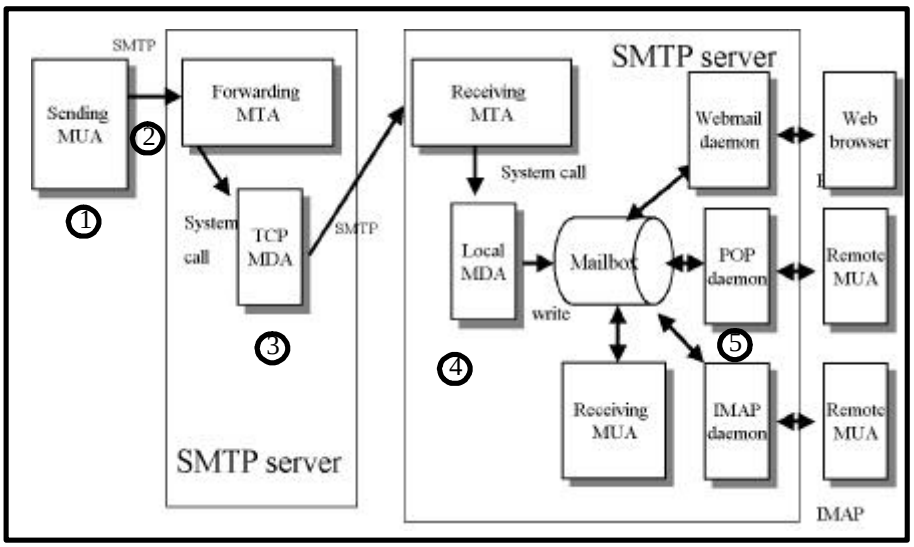

그림1 전자우편의 전달과정[6]

첫째 과정은 전자우편을 보내는 송신자가 'Outlook Express'같은 MUA(Mail User Agent)프로그램(그림1에서 좌측에 있는 'sending MUA': 표시①)을 이용하여 전자우편을 작성하고 전달을 명령하는 과정이다. 여기서 MUA(Mail User Agent)란 사용자가 전자우편을 보내고 받는데 사용하는 클라이언트 프로그램을 말하며, 그 예로는 Microsoft사의 'Outlook Express', Netscape사의 'Netscape mail', UNIX상에서 사용하는 '/usr/ucb/mail'등이 있다[6].

두 번째 과정은 이렇게 전송 명령을 받은 MUA프로그램이 사전에 사용자가 지정한 SMTP(Simple Mail Transfer Protocol)서버에 있는 'sendmail'과 같은 MTA(Mail Transfer Agent)프로그램에게 SMTP프로토콜을 이용하여 전자우편을 전달하는 과정이다 (표시②). 여기서 MTA(Mail Transfer Agent)란 받은 전자우편을 MDA(Mail Delivery Agent)를 호출하여 다른 MTA로 전송하거나 사용자의 우편함에 전달되도록 지시하는 프로그램을 말하며[6], 그 예로는 'sendmail'[5], 'qmail'등이 있다. 그리고 SMTP (Simple Mail Transfer Protocol)[7]는 MTA와의 통신을 위한 규약이다.

세 번째 과정은 전자우편을 받은 MTA프로그램이 수신자의 주소를 살펴보고 자신이 관리하고 있는 수신자가 아니면, 해당 서버에 전달하기 위해 TCP MDA(Mail Delivery Agent)를 호출하는(TCP network의 경우) 과정이다. 그러면 TCP MDA는 수신자가 있는 SMTP서버에서 동작하고 있는 MTA프로그램에게 역시 SMTP프로토콜을 이용하여 전달하게 된다 (표시③). 여기서 MDA (Mail Delivery Agent)란 MTA가 받은 전자우편을 전달하기 위해 사용하는 프로그램으로 외부로 전달하는 경우 채널(TCP, uucp 등)에 따라 각각의 MDA가 존재하며 '/bin/sh'에 의해 구동된다[6].

네 번째 과정은 세 번째와 같은 과정을 반복하여 최종적으로 수신자가 있는 SMTP서버에 있는 MTA프로그램이 전자우편을 받은 경우에 local MDA를 호출하여 사용자의 우편함에 배달하도록 지시하는 과정이다 (표시④). local MDA란 사용자의 우편함에 전자우편을 배달하는 MDA를 말하며, 그 예로는 'procmail', '/bin/mail'등의 프로그램이 있다. 우편함(mailbox)은 전자우편을 저장하기 위한 포맷으로 하나의 파일에 계속해서 덧붙이는 방식 (mbox)과 하나의 디렉토리에 각각의 전자우편을 각각의 파일로 저장하는 방식(MH, maildir)의 두 가지가 있다[6].

마지막 다섯 번째 과정은 수신자가 자신의 전자우편계정이 있는 SMTP서버에 직접 로그인 해서 유닉스의 'mail'과 같은 MUA프로그램을 통해 전자우편을 읽거나, POP (Post Office Protocol), IMAP (Internet Message Access Protocol)등의 서비스를 이용하여 원격지로부터 전자우편을 읽는 과정이다 (표시⑤). 여기서 POP (Post Office Protocol)이란 전자우편 서버에 외부에서 접속하여 전자우편을 확인하는 경우 사용하는 프로토콜의 한 종류를 말하는데, 최신 버전은 POP3[8]이며 사용한지 오래되었으므로 지원하는 서버가 많다는 점을 제외한 장점은 없다[9][10]. IMAP (Internet Message Access Protocol)이란 전자우편 서버에 외부에서 접속하여 전자우편을 확인하는 경우 사용하는 프로토콜의 한 종류로, 최신 버전은 IMAP4[11]이며 POP에 비해 기능이 많지만 지원하는 서버가 적다는 단점이 있다[9][10].

이상과 같이 전자우편의 전달과정을 통해 전자우편 시스템에 대해 개략적으로 살펴보았다. 본 논문에서는 전자우편 서버의 송신 측면보다는 수신 측면에 중점을 두었으므로 위의 전달 과정 중에서 그림1의 점선으로 표시한 부분과 같이 전자우편을 우편함에 저장하는 과정을 개선하여 성능을 향상시키고자 한다. 또 최근에는 웹브라우저를 이용하여 인터넷이 가능한 곳이면 어디서든 전자우편을 읽고 쓸 수 있는 웹-메일 방식이 보편화되고 있으므로 웹-메일을 지원하도록 했다.

2.1.2 Sendmail 프로그램

전자우편 서버 시스템을 이해하고자 할 때 시스템에 대한 전반적인 이해와 더불어 'sendmail'프로그램에 대한 이해도 필요하다. 우선 전자우편 서버 시스템을 개발할 때 MTA에 중점을 두고 전자우편 서버 시스템을 구현하는 경우는 말할 것도 없고, 본 논문처럼 MDA나 우편함(mailbox)관리 부분에 중점을 두어 구현하는 경우에도 MTA가 전자우편을 넘겨주므로 MTA에 대한 이해가 필요하다는 점은 명확하다. 그리고 비록 'sendmail'프로그램은 수많은 버그와 보안상의 문제점이 발견되기는 했지만 그 동안 여러 번의 수정을 통해 문제점을 해결해왔고 제작 후 지금까지 15년 이상 전 세계적으로 대다수의 서버에서 사용되고 있는 MTA프로그램이므로 다른 새로운 MTA들보다도 그 안정성과 성능을 신뢰할 수 있는 MTA프로그램인 것도 사실이다[3]. 따라서 이와 같은 점에 근거할 때 전자우편 서버 시스템의 개발에는 'sendmail'프로그램에 대한 이해가 선행되어야 한다는 것을 알 수 있다. 여기서는 'sendmail'프로그램을 사용하는데 있어서의 장점과 단점, 그리고 그 동작구조를 간략히 살펴보고자 한다.

2.1.2.1 'sendmail'프로그램 사용의 장점

sendmail[5][12]프로그램을 사용할 경우 다음과 같은 세 가지 장점

이 있다는 것도 MTA로 'sendmail'을 사용하는 이유가 될 것이다. 첫째, 'sendmail'은 호환성에 중점을 두어 만든 프로그램으로 서로 다른 network간의 전자우편도 잘 처리할 수 있도록 되어 있으며 대부분의 UNIX, linux 시스템에서 동작하게 되어 있다. 이것은 서로 다른 사양의 PC들로 클러스터를 구성할 때는 상당히 큰 장점이 될 것이다. 둘째, sendmail은 환경설정파일인 'sendmail.cf'를 수정함으로써 상당히 세밀한 동작의 조정까지 가능하다. 코드를 고치지 않고도 sendmail의 동작을 제어할 수 있다는 점에서 sendmail과 연관된 시스템을 개발할 때 쉽게 원하는 동작을 얻을 수 있는 장점이 있다. 셋째, sendmail이 원래 MDA를 어느 것이나 사용할 수 있도록 한 유연성 있는 구조라는 점은 별도의 MDA를 제작하여 사용할 수 있게 해주므로 여러 가지 새로운 서버 시스템의 시도가 자유롭다는 장점이 있다.

2.1.2.2 'sendmail'프로그램 사용의 단점

다만 'sendmail'프로그램을 사용할 경우 temporary file, 과도한 process forking, lock-file의 사용으로 인한 성능저하가 나타난다는 것이 단점이다[4]. temporary file문제는 sendmail이 전자우편을 받았을 때 queue디렉토리에 잠깐 생성했다가 삭제하는 qf-, df- 파일들의 overhead에 관한 문제이다[13]. 또 과도한 process forking문제는 sendmail이 SMTP요청을 처리하기 위해서 계속 process fork를 하는데 thread로 처리하지 않고 process를 생성하는 overhead가 크기 때문에 발생하는 문제이다. 마지막으로 lock-file의 문제란 sendmail이 전자우편을 처리할 때 queue에 있는 qf-파일에 대해 lf-라는 lock파일을 생성하여 lock을 처리하는데 linux에서 이러한 파일의 생성과 삭제에는 많은 시간이 걸리기 때문에 발생하는 문제이다. 이 밖에도 sendmail.net[13]에서 자체적으로 지적하고 있는 문제들로는 queue디렉토리가 하나일 경우 디스크에서 병목이 생기는 문제와 DNS 자체의 성능 저하등이 있다.

2.1.2.3 'sendmail'프로그램의 동작구조[14]

마지막으로 'sendmail'의 동작에 대해서 살펴보겠다. 'sendmail'은 SMTP연결이 들어올 경우 일반적으로 자식 프로세스를 fork하여 전자우편을 받은 뒤 배달을 시도하다가 실패하는 경우에 'sendmail'자체의 queue 디렉토리에 저장한다.

load average는 지난 1분 동안 전자우편을 처리하기 위해 실행중인 process들의 평균 개수를 말하는데 sendmail은 현재의 load average(la)를 검사하여 그 값이 미리 설정해둔 QueueLA값(x)보다 크게 되는 경우 'msgpri > q / (la - x + 1)'의 식을 계산한다. 이 식을 만족하는 경우 sendmail은 사용자의 우편함에 배달하는 일을 중지하고 도착하는 모든 전자우편들을 queueing하려고 시도한다. 또한 미리 설정해둔 refuseLA값보다 load average(la)가 크게 되는 경우는 더 이상의 SMTP 접속을 거부한다. 여기서 QueueFactor값(q)은 default 600,000으로 QueueLA값(x)은 default 8로 주어져있다. 또 msgpri는 message priority를 나타내는데 'msgpri = size - (class * z) + (receipients * y)'와 같이 계산한다. 여기서 size는 전자우편의 크기, receipients는 수신자의 숫자를 나타내며 class는 default 0, ClassFactor(z)는 default 1800, RecipientFactor(y)는 default 30,000이다. refuseLA는 default 12이다.

2.1.3 전자우편 서버 시스템에 관한 기존의 연구

여기서는 앞에서 살펴본 전자우편 시스템에 대한 설명과 'sendmail'프로그램에 대한 설명을 바탕으로 그 동안 제시되었던 전자우편 서버 시스템 관련연구들을 그 발전형태에 따라 순서대로 관련 연구들을 정리하고 분석해보려고 한다. 한 대의 기계로 구성한 전자우편 서버는 'sendmail'이 전자우편의 전송을 맡고 우편함의 관리는 local file system을 이용했다. 하지만 여러 대의 기계로 서버 시스템을 구성할 경우 이러한 방식은 사용할 수 없으므로 여러 가지 새로운 방식들이 제시되어왔다. 그 동안 제시되었던 방식으로는 'Cluster mailhub를 이용한 방식'[15], 'Duke 대학에서 사용한 방식

‘[1], ’EarthLink에서 사용한 방식‘[2], ’Porcupine Scalable Mail Server'[3] 등이 있다.

2.1.3.1 Cluster mailhub를 이용한 방식[15]

Cluster mailhub를 이용한 방식은 여러 대의 노드들이 NFS를 통해 하나의 mail spool에 접근할 수 있게 한 방법으로 scalable하지 않은 단점이 있었다. 그 한 가지 원인은 사용자 수가 많아짐에 따라 spool에 있는 file의 숫자가 증가하고 이것이 server의 directory name cache를 초과하는 경우 mailhub에 동작하고 있는 nfsd process에 걸리는 부하가 증가하므로 NFS접근이 느려진다는 것이다[1]. 또 한 가지의 원인은 SMTP 연결 숫자의 급증이다. 도착한 메시지는 sendmail이 자식 프로세스를 fork시켜서 처리하는데 앞에서 언급한 nfsd process의 load증가와 더불어, 동시에 들어오는 SMTP연결의 수가 급격히 증가하는 경우 가능한 처리 능력을 넘게 된다. 그러면 수행을 위해 기다리는 process의 수가 급증하므로 load average가[3)] 급증하고 sendmail은 load average를 낮추기 위해 요청 들어오는 SMTP connection을 거부하게 된다. 그러다가 기다리던 process들이 처리를 마치고 줄어들어서 다시 load average가 충분히 작은 값으로 떨어지게 되면 다시 sendmail이 SMTP 연결을 받아들이게 되는데 이처럼 들어오는 SMTP 연결이 집중될 때 문제가 발생할 수 있다[1].

2.1.3.2 Duke 대학의 사례[1]

Duke대학에서 제시한 방식은 [15]에서 제시한 mailhub방식에 mail gateway system을 설치하여 문제를 해결하려고 한 방식이다. 즉 DNS MX record를 수정하여 모든 전자우편을 mail gateway를 거치도록 조정한 것이다. 이러한 결과로 모든 SMTP 연결이 mail gateway를 통해 mailhub에게 직렬화 되어 전달되었다. 하지만

3) 본 논문의 ‘2.1.2.3 sendmail프로그램의 동작구조’를 참고

mailhub의 처리 속도보다 mail gateway의 queue에 들어오는 메시지의 속도가 훨씬 빨랐으므로 DNS MX record round robin을 이용하여 multiple gateway를 사용했다.

하지만 규모가 커짐에 따라 nfsd process간의 경쟁으로 인해 DISK I/O가 많은 시간이 걸리는 문제가 발생하였으므로 이 문제를 해결하기 위한 여러 가지 노력을 시도했으나 실패하고 우편함 접근을 위해 POP과 IMAP을 사용하도록 고쳤다. 즉 multiple post office machine을 두고 대용량의 CNAME DNS record와 sendmail의 userdb를 이용하는 방법을 택한 것이다. 사용자에게 각각의 mail server를 CNAME record를 이용하여 할당하고 자신의 mail software에서 그것에 접속하도록 한 뒤 시스템 변동 시에는 DNS record만을 수정하는 방법이다. 오늘날의 BIND[16] 프로그램이 많은 수의 CNAME record를 처리 가능한 점을 이용한 방법으로 사용자가 자신의 MUA프로그램에서 SMTP서버의 이름만 바꾸면 되므로 사용자가 해야할 일이 간단하다는 장점이 있다.

이 방식은 실제적인 대용량 전자우편 서버 시스템을 구성하기 위한 거의 최초의 시도이다. 특이한 점은 새로운 프로그램을 작성하거나 기존의 코드를 고치는 작업 없이 기존의 프로그램을 이용하면서 시스템구성을 변경하거나 환경설정을 조절하여 문제를 해결하려고 했다는 것이다.

2.1.3.3 EarthLink 방식.[2]

EarthLink사에서 제안한 방식은 [1]의 문제점을 개선하기 위해 제안한 방식으로 SMTP서버, POP서버, File서버, Authentication DB 서버로 구성되어 있으며 DNS MX 레코드의 Round Robin을 이용해 메시지를 분배했다. SMTP서버는 sendmail을 수정 없이 사용했으며 인증과 사용자 이름의 탐색을 시도하지 않도록 하여 성능향상을 꾀했다. MDA는 sendmail이 배포하는 mail.local을 수정하여 사용했다. 이것은 크기가 작고 몇 년 동안 거의 변화하지 않았으며 인증과 mailbox의 위치 확인작업 외에도 quota check도 할 수 있으므로 유

리하기 때문이다. 확장성을 위해 dataless POP서버를 사용했으나 POP daemon은 인증과 mailbox location정보를 알려주기 위해 상당히 기존의 프로그램에 큰 코드수정이 있었다. File서버는 메시지를 NFS같은 분산 파일 시스템에서 각 노드에 있는 디렉토리를 '/var/mail#'의 형태로 mount하여 사용하였고, 한 디렉토리 안에 너무 많은 수의 파일이 있지 않도록 우편함을 balanced hash로 분리하여 유지했다. 또한 mail queue를 유지하며 전자우편 서비스와 관련된 부가 파일들도 저장한다. Authentication DB 서버는 passwd file의 제약을 피하고 다른 응용 프로그램에서 정보를 이용할 수 있도록 SQL database내에 passwd file을 넣고 MDA내에서 getpwnam()대신에 SQL query를 수행하도록 하는 방법을 사용하고 있으며 이 외에 mailbox의 저장위치, 부가 파일들에 대한 정보를 갖고 있다.

이 방식의 장점은 인증을 한 곳으로 집중했으므로 동기화가 필요없고, 작은 OS 사용으로 문제 발생을 최소화하고, dataless한 POP 서버를 이용하여 교체가 용이하다는 점이다. 또 switched FDDI의 이용, dataless POP server의 이용, 이동 가능한 우편함을 통해 확장성도 갖추고 있다. 가용성을 위해서는 RAID를 쓰고 여분의 하드웨어를 준비해서 사용하고 있다.

반면에 다음과 같은 단점이 있다. 먼저 서버가 정지하는 경우 재빨리 DNS round robin에서 그것을 제거해야 하는 단점이 있다. 그리고 file locking의 경우 open system call의 O_EXCL flag를 이용했는데 lock이 가능하다는 것을 파악하는데 최대 15의 지연시간이 걸릴 수 있고 synchronous NFS operation이 크게 증가하는 단점이 있다. 또 저장공간과 사용자 DB서버를 위한 많은 수의 전용 기계들을 필요로 하고, 고가용성과 고안정성을 갖춘 하드웨어의 구입 비용이 많이 드는 단점이 있다. 또 구조가 고정되어 있으므로 파일서버 노드의 추가와 제거는 사용자 DB의 변화를 필요로 하고 메시지의 대량 이동을 초래한다. 마지막으로 SMTP와 POP 서버는 파일서버의 파일 시스템에 있는 lock file을 사용하는데 파일 생성과 삭제는 알려져 있다시피 느리기 때문에 오래 걸린다[3].

성능은 35만명의 사용자에 대해 460000개의 mailbox를 유지하고 있으며 초당 평균 20개의 메시지를 처리하는 셈이다. POP의 경우는 초당 20개의 새로운 POP connection이 가능하며 최대 600개의 daemon이 동작할 수 있으며 최대 1000개의 POP connection을 처리할 수 있다.

이 방식은 사용자 수에 대한 제한을 해결할 수 있도록 사용자 DB를 사용하도록 제안했고 spool에 hash개념을 적용해서 탐색시간을 줄였으며 우편함을 확장할 수 있게 했다는 점에서 의미가 있다. 하지만 여전히 NFS방식의 한계를 안고 있으며 가용성을 위해 여분의 구성 요소를 필요로 한다는 점과 인증서버, 파일서버같이 핵심적인 부분이 여전히 병목이 될 수 있다는 점에서 문제점이 남아있다.

2.1.3.4 Porcupine Scalable Mail Server [3]

Porcupine Scalable Mail Server는 DNS의 A, MX record를 이용하여 SMTP, POP, IMAP socket들을 client에게 제공하는 방식이다. 각 노드간의 역할 분담이 없고 각각 독립적으로도 기능 할 수 있도록 한 노드에서 SMTP, POP, IMAP session을 처리할 수 있고 전자우편의 저장과 사용자 DB관리도 할 수 있게 되어있다. 또한 사용자 집합의 부분집합을 유지하고 있으며 사용자의 우편함은 노드들에 분산되어 있다. 각 노드는 사용자 우편함의 부분을 저장할 수 있으며 사용자의 전자우편은 사용자가 POP이나 IMAP으로 우편함을 읽으려고 할 때 모아서 제공한다. 사용자의 전자우편은 어느 노드에나 있을 수 있으며 어디에 흩어져 있는지 나타내는 노드들의 정보를 따로 유지하고 있다.

사용자 데이터베이스는 user directory라는 각 노드에 모두 복사되어 있는 메모리상의 작은 테이블을 사용하는데 사용자 이름을 사용자 노드로 대응시키는 정보를 담고 있다. 노드가 동작을 중지했을 경우는 distributed membership protocol을 이용해서 새로운 user directory를 배포한다.

전자우편이 도착한 경우 사용자 DB를 참조하여 그 사용자가 유효

한 지 확인한 다음 network이동을 줄이는 방향으로 메시지를 저장할 노드를 선택하고 거기에 저장한 뒤 사용자 DB에 그 정보를 추가한다. 사용자가 전자우편을 읽으려고 시도하는 경우는 사용자 DB에 접근하여 우편함의 정보를 얻어내고 parallel RPC를 이용하여 메시지를 가져온다.

그리고 분산되어 흩어져있는 메일박스와 user directory정보를 관리하기 위하여 NFS gateway daemon을 이용했다. 이것은 하나의 directory에 관련 파일들이 모두 있는 것처럼 보여주므로 시스템을 조사하고 변경하는 표준적인 OS 도구들을 사용할 수 있게 한다.

이 방식의 장점은 노드가 동작을 정지하면 거기에 있던 메시지는 읽을 수 없지만 그 이후에 도착하는 메시지들을 계속 받을 수 있도록 했다는 것이다. 이전의 방식은 노드가 죽으면 그 노드에 있는 사용자들은 메시지를 받을 수 없었다. 또 노드간의 구분이 없으므로 부하분산이 동적으로 이루어질 수 있고, 노드를 삭제했을 때는 DNS만 변경하면 된다는 장점이 있다. 마지막으로 mailing list는 전자우편의 상당한 부분을 차지하는데 이것을 하나만 두고 나머지는 link를 둠으로써 효율적으로 자원을 사용할 수 있다는 장점이 있다.

Myrinet을 사용했고 linux에서 pthread를 기반으로 개발했으나 아직 개발 중이므로 prototype에 대한 simulation으로 성능을 측정했다. 시스템이 완전히 scalable하다고 가정할 때 계산상으로 하루에 10억 개의 메시지를 처리하려면 650개의 노드가 필요하다고 했지만 이 때 network용량은 무한하다고 가정했으므로 계산의 의미는 없다. 실제 이렇게 되려면 양방향 1Gb/sec의 bandwidth가 650개의 node사이에 필요하기 때문이다. 자체적으로는 Disk가 가장 큰 bottleneck으로 파악하여 빠른 disk를 사용할 경우 성능의 증가가 있다고 보았다.

이 방식은 NFS에 의존하던 기존의 방식을 탈피함으로 해서 NFS 때문에 발생하던 문제들을 피했다는 장점은 있지만 broadcast를 비롯해서 곳곳에 흩어져 있는 우편함을 모을 때 일어나는 엄청난 양의 network traffic이 단점이다.

2.2 클러스터 구성 방식에 관한 기존의 연구

클러스터 구성에 대한 기존의 연구들을 살펴보기 전에 먼저 클러스터에 대해 간략히 살펴보겠다. 클러스터는 단일 리소스(시스템)로 사용되기 위해 네트웍으로 연결된 독립된 컴퓨터들의 집합이다. 클러스터는 사용되는 용도에 따라 계산 클러스터링과 IP 클러스터링으로 구분할 수 있다. 계산 클러스터링은 기존의 수퍼 컴퓨터를 대체하기 위한 대안으로써 고성능(High Performance)과 확장성(Scalability)을 주된 목표로 한다. 반면, IP 클러스터링은 기존의 기업용 서버에 대한 대안으로써 고성능과 고가용성(High Availability), 확장성(Scalability) 등을 목표로 한다. 지금까지 IP 클러스터링에 관한 연구로 주로 웹 서버에 집중되어 있었다. 이러한 지식을 바탕으로 여기서는 기존의 웹 서버 클러스터링을 통해 이루어진 고성능의 시스템을 구성하기 위한 다양한 기법을 살펴보겠다.

웹 서버는 HTTP 연결 설정을 위한 빈번한 요청을 처리해야 한다. 따라서, 이러한 요청을 효과적으로 처리하기 위해서는 클러스터 노드에 요청을 적절하게 분배하여 시스템 전체의 처리율을 높이는 기법이 필요하다. 이러한 기법으로는 Round Robin DNS(RRDNS)[17], Network Address Translation(NAT)[18], Tunneling[19], Direct Routing(DR)[19], Broadcast & Filtering(BF) 등이 있다.

2.2.1 Round Robin DNS(RRDNS)

Round Robin DNS(Domain Name Service) 기법은 NCST 웹 서버에 처음 도입되어 지금까지 널리 사용되고 있는 부하 분산 기법으로 가장 쉽게 클러스터를 구성할 수 있다는 장점이 있다. DNS는 클라이언트가 요청하는 도메인 이름에 해당 IP 주소를 알려주는 역할을 한다. RRDNS는 하나의 도메인 이름에 여러 개의 IP를 할당하고, 매 클라이언트 요청에 대하여 이들 IP 중에서 이전 요청과 다른 IP를 되돌려 줌으로써 부하조절을 하는 기법이다. 현재 BIND(Berkeley Internet Name Domain) 프로그램이 DNS 서버로

널리 사용되고 있는데, 이 프로그램에는 RRDNS기능이 이미 구현되어 있다.

이 기법의 장점은 설정 파일 수정만으로 클러스터를 구성할 수 있고, 클라이언트가 처음 도메인 이름에 질의를 할 때에만 DNS 트래픽이 발생함으로 특정 서버에 병목이 생기지 않는다는 것이다. 그러나, 이 방식은 DNS 서버가 클러스터 노드의 상태를 알 수 없기 때문에, 노드가 동작하지 않는 경우에도 유효하지 않는 IP를 계속 제공하게 되어 일부 사용자에게 서비스가 되지 않는 문제를 발생시킨다. 또한, DNS 서버는 계층 구조를 이루고 있기 때문에 RRDNS 서버 이외의 DNS 서버에 대한 가정을 할 수 없다. 즉, 다른 DNS 서버가 성능을 높이기 위해 캐쉬를 하는 경우, 부하는 특정 노드로 집중될 수 있다.

2.2.2 Network Address Translation(NAT)

Network Address Translation(NAT) 기법은 IP 패킷 내의 주소를 변경하여 부하 분산을 수행하는 기법이다. 클라이언트는 가상 서버의 도메인 이름과 IP 주소, 이 예에서는 부하 조절자(Load Balancer)의 도메인 이름과 IP 주소로써 서버에 접속을 시도한다. 요청을 받은 부하 조절자는 적절한 스케줄링 기법을 통해 패킷을 전달한 노드를 선택하고, 해당 노드의 IP 주소로 받은 패킷의 목적지 주소를 변경한다. 이 때, 부하 조절자는 check sum 등 IP 주소 변경으로 인한 부가 작업을 수행해야 한다. 이어, 패킷이 해당 노드에 도착하면, 그 노드는 요청을 처리한 후에 부하 조절자에게 응답을 되돌려 준다. 응답 패킷을 받으면 부하 조절자는 응답의 발신지 주소를 자신의 IP로 고쳐 쓴 후 클라이언트에게 재전송한다.

이 기법은 적절한 오류 전달(Fail-over)기법을 사용하는 경우 클러스터 노드의 중단을 감지할 수 있고, 스케줄링 대상에서 제외할 수 있으므로 RRDNS와 같이 서비스가 중단되는 단점을 보완할 수 있다. 또한, 클러스터 노드에 사설 IP를 할당하고 부하 조절자에만 실제 IP를 할당하므로 IP 등록 비용을 절약할 수도 있다. 그러나, 부하

조절자가 모든 패킷에 대한 조작을 해야하므로 병목이 되어 구성할 수 있는 노드의 수가 제약을 받으며, 부하 조절자가 중단되는 경우에는 서비스가 중단되는 문제점이 있다.

2.2.3 Tunneling

Tunneling 기법은 IP 데이터 그램을 새로운 IP 데이터 그램으로 감싸므로써 특정 IP로 전송된 패킷을 다른 IP로 전송할 수 있게 하는 기법이다. 부하 조절자가 클라이언트로부터 온 패킷을 받으면 부하 조절자는 스케줄링에 의해 선택된 노드로 수신한 패킷을 새로운 IP 데이터그램 패킷에 실어 전달한다. 해당 노드는 수신한 패킷으로부터 원래 패킷을 추출하고 요청한 서비스를 처리한 후 직접 클라이언트에게 응답을 보낸다.

이 기법은 모든 노드가 패킷을 열어 처리할 수 있는 루틴이 구현되어야 한다는 점과 패킷을 새롭게 구성하고 추출하는데 부담이 있다. 그러나, CRC 검사, 주소 변경 등 수신한 패킷에 대한 별도의 조작이 없이 내부 네트웍으로 패킷을 전송할 수 있다는 장점이 있다.

3.2.4 Direct Routing(DR)

하나의 네트웍 디바이스에 실제 IP 이외에 여러 개의 가상 IP를 할당할 수 있다. Direct Routing(DR) 기법은 노드와 부하 조절자에 동일한 가상 IP를 부여하고, 부하 조절자가 요청 패킷을 스케줄링 기법을 사용하여 선택된 노드로 패킷을 직접 전송하는 기법이다. 우선, 클라이언트가 가상 IP를 통해 서비스를 요청하면 부하 조절자는 적절한 스케줄링을 통해 노드를 선택한다. 부하 조절자는 패킷의 MAC 주소값을 해당 노드의 Ethernet 주소값으로 변경하여 해당 노드로 직접 전달한다. 패킷을 전달받은 노드는 요청을 처리한 후 직접 클라이언트에게 응답을 전송한다. 각 노드는 부하 조절자와 동일한 가상 IP를 공유하고 있기 때문에 부하 조절자를 거치지 않고 클라이언트에게 직접 응답을 보낼 수 있다.

이 방식은 응답이 부하조절자를 거치지 않기 때문에 NAT 기법보다는 부하 조절자가 병목이 될 가능성이 줄어든다. 그러나, 이 기법은 모든 노드가 실제 IP를 가지고 있어야 하고, 부하 조절자의 중단으로 인해 문제를 해결해야 하는 단점이 있다.

3.2.5 Broadcasting & Filtering (BF)

Broadcasting & Filtering(BF) 기법은 앞서 설명한 NAT, tunneling, DR 기법에서의 부하 조절자로 인한 병목 현상을 제거한다. 즉, 클라이언트는 가상 IP를 통해 서비스를 요청하고, 패킷을 받은 라우터는 이더넷 방송 주소(FF:FF:FF:FF:FF:FF)를 사용해 클러스터 내의 모든 노드에 방송을 한다. 패킷을 전송받은 각 노드는 필터링을 하여 자신이 처리해야하는 패킷인 지 판단하여, 처리해야 하는 패킷이면 처리하고 결과를 돌려주고 그렇지 않은 경우에는 버린다. 각 노드는 가상 IP와 실제 IP를 가지고 있고, 별도의 노드 ID를 가지고 있어야 한다. 각 노드는 수신한 패킷의 발신자 주소와 포트 번호에 해쉬를 적용하여 자신의 ID와 같으면 처리한다.

이 기법은 부하 조절자가 없으므로 병목 현상이 발생하지 않고, 추가로 중복된 부하 조절자를 필요치도 않는다. 비록, 이더넷 방송으로 인해 모든 노드가 트래픽 증가 부담을 겪지만, 서비스 요청이 인바운드 트래픽이 적고 아웃바이드 트래픽이 매우 많은 일반적인 웹 부하의 측면을 고려할 때 방송에 의한 부담은 그리 크지 않은 것으로 알려져 있다.

3. 구현

본 논문에서 구현한 웹-메일 서버 시스템은 리눅스 PC 클러스터를 이용하고 있으며 특히 클러스터의 확장성을 이용하여 전자우편 서버 시스템의 성능 중에서도 수신 성능의 향상에 초점을 맞추고 있다. 여기서는 앞의 관련 연구의 내용을 바탕으로 시스템이 실제 전자우편을 수신하고 사용자가 웹-메일 서비스를 이용해 이 전자우편을 확인하는 과정까지의 예를 통해 자세한 설명을 한다. 특히 기존의 시스템과 다르게 구현한 우편함(mailbox) 관리부분, 웹-메일 서비스 지원을 위한 부분들을 중점적으로 설명한다.

3.1 전자우편의 수신

웹-메일 서버 시스템의 구성은 그림2와 같이 웹-서버와 전자우편 서버로 되어 있으며 전자우편의 수신은 이 중에서 전자우편 서버만 관여한다. 그림2의 오른쪽 하단에 보면 외부에서 온 전자우편이 도착하는 것을 나타낸 화살표를 볼 수 있다. 이것은 관련 연구에서 소개한 일반적인 전자우편 전달 과정에 따라 웹-메일 서버 시스템의

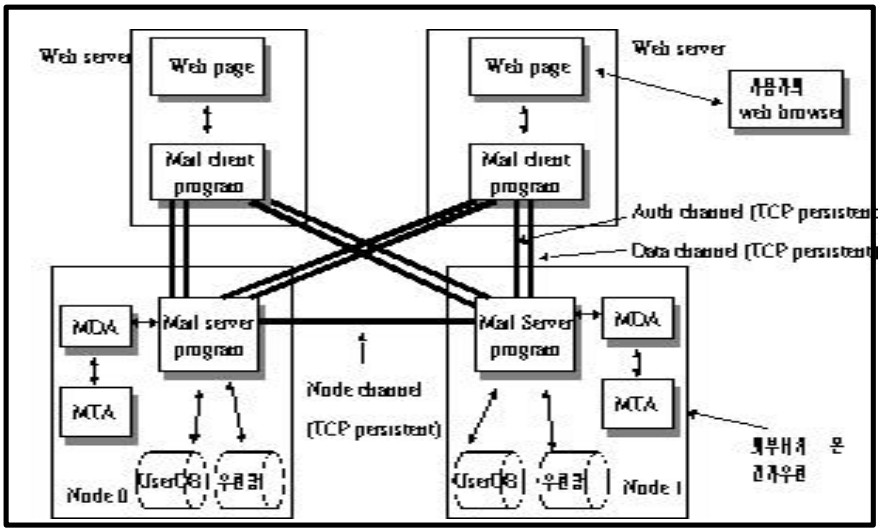

그림2 웹-메일 시스템의 구조

노드에 있는 MTA에 전자우편이 도착한 상황을 나타낸다.

3.1.1 MTA(Mail Transfer Agent)프로그램

여기서 사용하고 있는 MTA프로그램은 'sendmail'프로그램으로 기존의 'sendmail'을 수정 없이 사용하고 있다. 왜냐하면 새로 MTA를 만드는 일은 그 자체만으로도 어렵고 비용이 많이 드는 일이기 때문이다. 또 새로 만들지 않고 단지 sendmail을 수정하는 경우라도 sendmail의 새로운 버전이 몇 달마다 발표되면 그 때 마다 거기에 다시 수정을 가해야 하는데 그것은 자체 MTA를 만들고 유지하는 것만큼 힘든 일이기 때문이다[2].

전자우편을 받은 'sendmail'은 forwarding을 처리하기 위해 '/etc/passwd' 파일을 참조하지만 처리 속도를 높이기 위해서 forwarding을 생략하도록 환경설정을 바꾸었다. 이것은 'sendmail.cf' 파일에서 'Mlocal'정의 부분의 'F=w' flag를 제거함으로써 가능하다[2].

3.1.2 MDA(Mail Delivery Agent)프로그램

이러한 처리가 끝나면 'sendmail'은 받은 전자우편을 처리하기 위해 프로세스를 fork하여 local MDA를 호출한다. 보통의 linux시스템은 local MDA로 'procmail'프로그램을 사용하지만 웹-메일 시스템에서는 사용자 수의 제약 때문에 사용할 수 없다. 'procmail'프로그램은 수신자의 존재여부를 '/etc/passwd' 파일에서 확인하는데 여기에 30,000명에서 65,000명의 사용자를 추가하는 경우 제공업체에 따라 그 동작을 예측할 수 없다. 그것은 system에 따라 사용 가능한 UID의 개수가 다르기 때문이다[2]. 이러한 사용자 수의 제약은 많은 사용자에게 서비스를 제공해야 하는 대용량 웹-메일 서버에는 치명적이므로 'procmail'을 사용할 수 없다. 본 논문에서는 [2]에서 접근한 것처럼 'sendmail'[18]에서 배포하는 'mail.local'프로그램을 수정하여 사용했다. 현재 수신자가 시스템 내에 존재하는 사용자인지 확

인하는 부분은 없으며 단순히 'sendmail'로부터 전달받은 전자우편을 'mail server program'에게 TCP 연결을 이용해 전송하는 역할만을 담당한다.

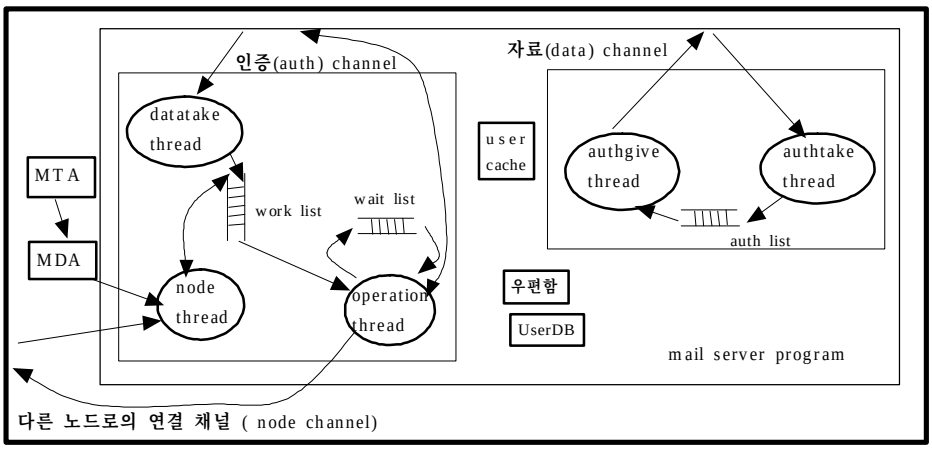

그림3 전자우편 서버 노드의 구조

3.1.3 mail server program

'mail server program'은 각각 독립적으로 처리해야 하는 모듈들을 POSIX pthread를 이용하여 다중 쓰레드로 구현했는데 프로세스를 fork하는 것보다 작업 전환시의 overhead가 적기 때문이다. 웹-서버로부터 사용자 인증에 대한 요청을 받아들이는 'authtake thread', 그 요청을 처리하고 그 결과를 웹-서버에 돌려주는 'authgive thread'는 인증 부분을 구성한다. 웹-서버로부터 작업 처리 요청을 받아들이는 'datatake thread', 실제로 작업을 처리하는 'operation thread', 노드 간의 통신과 MDA와의 통신을 담당하는 'node thread'는 작업 처리 부분을 구성한다.

3.1.3.1 node thread

'mail server program'가 local MDA로부터 전자우편을 받는 경우 'node thread'가 처리하도록 한다. 'node thread'는 'node channel'이

라는 TCP persistent 연결을 통해 노드간의 통신을 담당하며, local MDA로부터 받은 전자우편을 처리하는 일도 담당한다. 'node thread'는 이렇게 도착한 전자우편을 실제 작업 처리를 담당하는 'operation thread'가 내부에서 처리할 수 있는 형식으로 변환하여 대기 queue에 집어넣는다. 이 대기 queue의 이름은 'work list'이며 'operation thread'가 인출해서 처리할 때까지 임시 저장소 역할을 한다. 그림3은 이러한 과정을 나타낸 것이며 그림2의 'mail server program' 부분을 자세히 나타낸 것이기도 하다. 또 'work list'는 웹-서버에서 오는 웹-서비스 관련 작업의 대기 queue 역할도 함께 한다.

3.1.3.2 operation thread

'work list'에서 처리할 작업을 인출해온 'operation thread'는 먼저 수신자가 시스템에 존재하는지 확인한다. 시스템에서 사용하는 수신자 혹은 사용자 인증은 앞에서 언급한 바와 같이 사용자 수의 제약 때문에 '/etc/passwd' 파일을 사용하지 않는다. Berkeley-style NEWDB를 이용한 hashed passwd file을 사용하는 방법은 매번 hash를 rebuild해야 하므로 적합치 않으며[2], SQL DB에 사용자 인증을 위한 정보를 유지하는 방법[2]은 DB에 접근하는 시간이 너무 오래 걸려서 사용하기에 적당하지 않다. 그림3의 user cache는 요청이 많은 일정 숫자의 사용자들에 대해서만 사용자의 ID, 패스워드, primary node number, secondary node number만을 메모리 상에 올려서 인증에 사용하는 자료 구조이다. 이와 같은 방법은 관련 연구에서 살펴본 Porcupine[3][4]의 방식과 비슷하지만 사용자 인증 정보를 모두 메모리 상에 두지 않고 일부 접근이 많은 사용들에 대해서만 유지하므로 메모리 사용이 그만큼 적은 장점이 있다.

'operation thread'가 메모리의 자료구조로부터 사용자가 존재하는 것을 확인하면 사용자의 primary node number를 확인한다. primary node number는 사용자의 우편함과 사용자의 웹-서비스 관련 정보를 유지하고 있는 노드의 번호이며 secondary node number는

primary node에 문제가 생겼을 때 그 역할을 대신할 노드의 번호이다. 사용자 인증에 필요한 정보와는 달리 이름, 주소, 나이처럼 웹-서비스를 위해서만 필요한 사용자의 웹-서비스 관련 정보는 사용자 정보의 등록, 갱신, 삭제와 같이 드물게 발생하는 작업에만 필요하므로 SQL DB에 유지하도록 한다.

만약 수신자의 primary node가 현재의 노드라면 사용자의 우편함에 배달을 시도하고 아니라면 해당 노드에게 node channel을 통해 처리를 요청한다. 이렇게 다른 노드에게 작업 처리를 요청한 경우에는 'wait list'라는 별도의 대기 queue에 이 작업에 대한 정보를 유지하고 나중에 그 노드로부터 응답이 오면 다시 인출하여 작업을 마무리한다. 사용자의 우편함은 기본적으로 UNIX나 linux에서 사용하는 mbox 형식을 사용한다. 이것은 '사용자ID'와 같은 이름의 파일에 사용자의 전자우편을 계속해서 덧붙여 가는 방식으로 보통의 MUA프로그램을 이용해서 전자우편을 볼 수 있고 추후에 POP같은 서비스를 지원하기 쉽다는 장점이 있기 때문이다. 사용자마다 directory를 생성하고 전자우편마다 하나의 파일로 유지하는 방법도 가능하지만 그러한 경우 한 노드에 32,000명 이상의 사용자를 서비스할 수 없는 문제가 발생한다. Redhat 6.2 linux의 경우 한 directory아래 sub-directory를 32,000개 이상 만들 수 없기 때문이다. 웹-메일 서비스의 경우 보낸 편지에 대한 정보도 유지해야 하므로 그런 전자우편의 경우 '사용자ID.sent'라는 이름의 파일에 mbox 형식으로 저장하여 따로 관리한다. 또 전자우편의 개수가 많은 경우 그 요약정보를 얻기 위해 크기가 큰 mbox 파일을 검색하는 것은 시간이 오래 걸리므로 이러한 작업을 빠르게 하기 위해 각 전자우편의 header들을 따로 '사용자ID.info'라는 이름의 파일에 유지하도록 한다. 전자우편의 요약정보는 사용자이름, 제목, 상대방주소, 처리한 날짜, 크기, 상태 (읽었는지 여부 등) 등을 포함한다. 이러한 우편함 관리 방식은 사용자에 대한 전자우편이 한 노드에 모두 모이므로 관리하기가 쉽고 Porcupine[3][4]방식과 달리 사용자가 전자우편을 읽을 때 발생하는 과도한 network overhead가 없다는 장점이 있다.

3.2 웹-서비스를 통한 전자우편의 확인

앞에서 설명한 것처럼 전자우편이 도착하고 나면 사용자는 웹브라우져를 이용해 전자우편을 확인할 수 있게 된다. 사용자가 웹-메일 시스템을 이용하기 위해서는 먼저 웹브라우져를 이용하여 웹-메일 서버 시스템에 로그인 해야한다. 사용자가 웹페이지에 있는 인터페이스에 자신의 ID와 패스워드를 입력하면 웹-서버는 script언어를 구동하여 웹-서버에서 동작하고 있는 'mail client program'에 TCP 연결을 요청한다.

3.2.1 웹-서버

웹-서버는 기존의 일반적인 웹-서버를 사용하지만 웹-메일 서비스를 지원하기 위해 전자우편 서버와의 연결을 중계하는 'mail client program'을 사용한다는 점이 일반적인 웹-메일 시스템과 다르다. 일반적인 웹-메일 시스템은 script언어로 전자우편 서버에 직접 접근하지만 본 논문에서는 'mail client program'을 통해 전자우편 서버에 연결하게 함으로써 전자우편 서버의 구성이 변하더라도 웹-서버 쪽의 부분은 영향을 받지 않게 한다. 이것은 전자우편 서버를 클러스터로 구성한 것과 상관없이 기존의 웹-서버를 그대로 이용할 수 있게 한다.

3.2.2 mail client program

'mail client program'은 동시에 여러 명의 사용자에게 서비스를 제공하기 위해 여러 개의 쓰레드들로 쓰레드풀(thread pool)을 구성하고 사용자가 요청하는 매 작업마다 쓰레드를 할당하고 반납하도록 한다. 그림4는 이러한 과정을 나타낸 것으로 그림2의 'mail client program' 부분을 더 자세하게 나타낸 것이다. 웹-서버 쪽과는 매 작업마다 다시 연결을 시도하지만 전자우편 서버와는 사용자 인증

을 위한 채널 (auth channel)과 작업 처리를 위한 채널 (data channel)의 두 채널을 TCP 연결로 계속 유지하도록 해서 매번 TCP 연결을 설정하고 해제하는 overhead를 피한다.

웹-서버가 script를 통해 사용자 인증을 요청하면 웹-서버에 있는 'mail client program'은 TCP 연결을 통해 이 요청을 받아서 임의의 클러스터 노드에게 중계해준다. 사용자의 인증이 끝난 다음에는 사용자의 primary node로 연결을 설정하여 data channel을 통해 작업 관련 정보들을 주고받는 중계 역할을 한다.

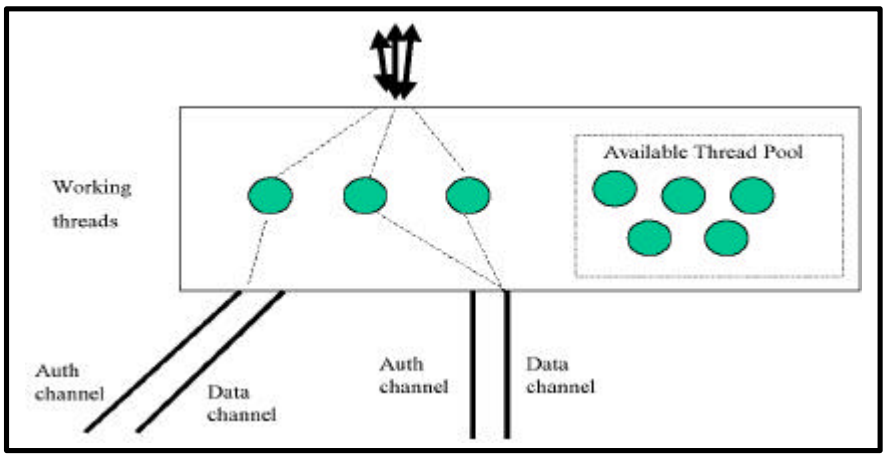

그림4 mail client program의 구조

3.2.3 authtake thread, authgive thread

'mail client program'으로부터 auth channel을 통해 인증 요청을 받으면 'authtake thread'는 'auth list'라는 대기 queue에 요청 작업을 저장한다. 이와는 반대로 'authgive thread'는 대기중인 작업 요청을 'auth lsit'로부터 인출하여 앞에서 언급한 user cache에서 사용자의 ID와 패스워드를 비교함으로써 사용자 인증 결과를 'mail client program'에게 auth channel을 통해 돌려준다. 사용자 인증이 성공한 경우에는 사용자의 primary node 정보를 같이 돌려주는데 이 정보를 이용하여 'mail client program'은 사용자를 자신의 primary node로 연결시켜준다.

3.2.4 datatake thread

이상의 과정을 통해 사용자 인증에 성공한 사용자는 전자우편 관련 작업들을 할 수 있게 된다. 전자우편 관련 작업이란 전자우편을 보내고 읽고 지우는 등의 모든 작업들을 말하지만 본 논문은 전자우편 수신에 초점을 두고 있으므로 여기서는 읽는 작업에 대해서만 본다. 사용자의 작업 요청은 웹브라우져에서 'mail client program'을 통해 primary node의 data channel로 전송되며 'datatake thread'가 이것을 받아서 'operation thread'가 처리할 있는 형식으로 바꾼 다음 대기 queue인 'work list'에 저장한다.

3.2.5 operation thread

전자우편의 수신 과정을 통해 'operation thread'의 역할을 설명했듯이 'work list'라는 대기 queue로부터 작업 요청을 인출해서 처리하는 일을 한다. mbox형태로 저장되어 있는 전자우편을 읽어서 웹-서버 쪽의 script가 처리할 수 있도록 보내준다. 전자우편을 웹브라우져로 볼 수 있도록 가공하는 일은 웹 쪽의 script에게 맡긴다.

4. 실험

기존 시스템에서 수신 성능 저하의 가장 큰 원인이 되는 부분이 어느 곳인지를 확인하는 실험과 본 논문에서 구현한 시스템이 기존의 시스템에 비해 어떤지 평가하고 성능 개선을 위한 자료를 얻기 위해 성능 측정 실험이 필요하다. 특히 전자우편의 전달 과정 중에서 MDA(Mail Delivery Agent)로부터 사용자의 우편함(mailbox)까지의 과정을 수정한 시스템이므로 MTA를 비롯한 다른 조건들은 동일하게 설정한 기존의 시스템과 비교실험이 적합할 것이다. 본 장에서는 클러스터 구성, 네트워크 구성, 실험 기계의 사양과 같은 각종 실험 환경 설정을 설명하고 실험 내용을 본 뒤 실험 결과를 제시하고 분석한다.

4.1 실험환경

현재 클러스터를 구성한 방법은 관련 연구에서 소개했던 기법 중에서 round robin DNS방법[14]을 사용하고 있다. 이 방법은 노드가 문제를 일으켰을 때 제대로 처리할 수 없는 문제가 있는 방법이기는 하지만 본 논문의 초점이 가용성보다는 성능에 초점을 맞추고 있고 앞으로의 연구를 위한 프로토타입을 구현한다는 점에서 우선 실제로 많이 사용해왔고 설정하기 쉬운 방법을 선택하기로 했기 때문이다. round robin DNS방법을 사용하기 위해서는 DNS record를 자유롭게 수정할 수 있어야 하므로 외부 네트워크와는 독립된 별도의 네트워크를 구성하고 이 안에서만 사용하는 DNS서버[13]를 설치했다. 또 네트워크는 100Mbps Fast Ethernet switch를 기준으로 했는데 외부 네트워크의 역할을 할 switch와 내부 네트워크의 역할을 할 switch를 따로 두어 다른 기계에 의한 간섭을 피하고 각 노드들 간의 통신을 위한 독립된 채널을 확보하도록 했다. 이를 위해서 각 노드에는 네트워크 card를 2개씩 설치했다. 구성은 그림5와 같다.

각 노드들과 송신자들은 PentiumIII 550 MHz dual CPU, 256MB RAM, UDMA66 EIDE 5GB HDD의 사양으로 운영체제는 Redhat

Linux 6.2를 사용했다. 각 기계에서 동작하는 sendmail은 sendmail 8.11.1을 사용하였고 sendmail관련 설정의 변경은 'Mlocal'관련 부분만을 수정했다. local MDA를 'sendmail'에서 배포하는 'mail.local'로 설정하도록 하고 forwarding을 생략하도록 'F=w' flag를 제거했으며 round robin DNS를 통해 도착한 전자우편을 받도록 'Cw'항목을 수정했다. 그 이외의 설정은 default로 설정했다.

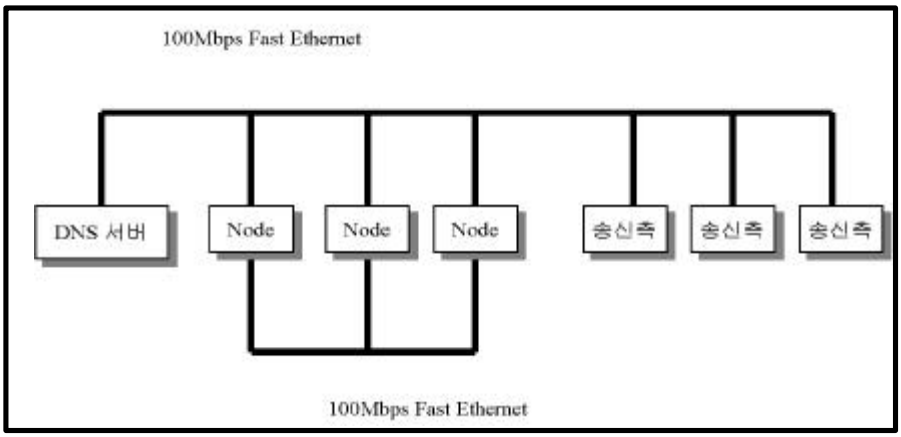

그림5 실험환경의 네트워크 구성

사용자의 수는 기계 한 대 당 30,000명으로 설정했다. 왜냐하면 '/etc/passwd' 파일을 사용하는 기존 시스템과의 비교를 하는 경우 앞에서 언급한 바와 같이 '/etc/passwd' 파일의 사용자 수가 35,000에서 60,000이 넘는 경우 문제가 발생할 수 있으므로[2] 이를 피하기 위해서이다.

전자우편의 크기는 512 Byte, 4 KByte, 32 KByte, 256 KByte의 네 가지 종류에 대해서 실험했다. 이것은 각각 간단한 내용의 메모, 비교적 장문의 전자우편, 문서파일을 포함한 전자우편, 그림파일을 포함한 전자우편의 크기와 비슷한 크기로 전자우편 시스템에서 처리하는 대표적인 전자우편의 크기라고 할 수 있기 때문이다.

4.2 실험내용

본 논문은 전자우편 서버의 성능 중에서도 수신 성능에 초점을 두었으므로 수신 성능에 대한 측정 실험을 한다.

기존 시스템의 수신 성능 저하 원인을 확인하기 위한 실험은 전자우편의 크기를 변화시켜가면서 수신 측에서 bottle nect현상이 일어날 때까지 전자우편을 집중적으로 송신하고 이 때 'sendmail'이 남긴 log를 분석하는 방법을 사용했다.

성능을 측정하기 위한 방법으로 기존 시스템의 경우는 'sendmail' 프로그램이 '/var/log/maillog'라는 파일에 남기는 log의 내용을 분석하고 'mailq' 프로그램을 이용했다. 1초 당 처리할 수 있는 최대 전자우편 수를 평가 기준으로 삼는다. 본 논문에서 구현한 시스템은 'sendmail'의 'log'파일을 사용할 수 없으므로 'mail server program'의 내부에서 전자우편을 받아서 처리를 완료할 때까지의 각 과정마다 'gettimeofday'함수를 이용해 시간 정보를 기록하고 최종적으로 시간을 계산하도록 한다. 역시 1초 당 처리할 수 있는 최대 전자우편 수를 평가 기준으로 삼지만 latency를 측정하여 그로부터 처리 가능한 수를 추정한다는 점이 다르다. 여기서 latency란 전자우편 한 통만을 수신하도록 하여 그 때 걸리는 시간을 말한다. 이렇게 하는 이유는 송신하는 측에서 초 당 송신하는 전자우편의 수를 증가시키면 디스크에 메시지를 쓰는 시간이 증가하여 한 통을 수신하는데 걸리는 평균시간이 증가하는 경향을 보이므로 정확한 측정이 어렵기 때문이다.

먼저 '/etc/passwd'파일과 'procmail'프로그램을 이용한 기존 시스템에서 수신 성능 저하의 원인이 되는 부분을 확인하기 위한 실험을 하고 이 시스템의 수신 성능 측정 실험을 한다. 다음으로 사용자 인증과 메시지 요약정보 관리를 postgres DB로 하고 메시지마다 파일을 따로 두어 저장하는 시스템에 대한 수신 성능 측정 실험을 한다. 마지막으로 본 논문에서 구현한 시스템에 대한 수신 성능 측정 실험을 수행한다. 특히 postgres DB를 이용한 시스템은 본 논문에서 구현한 시스템의 개발 단계에서 시도했던 접근 방식으로 현재의 시스템과 DB를 사용한 부분만 다른 시스템에 대한 실험을 통해 DB를 사용했을 때의 결과를 알 수 있게 한다.

4.3 실험결과

4.3.1 기존 시스템에서 수신 성능 저하의 원인 확인

'/etc/passwd'파일과 'procmail'프로그램을 이용한 기존 시스템의 경우 전자우편을 배달하는데 가장 시간이 많이 걸리는 부분은 MDA가 사용자의 우편함에 전자우편을 덧붙이는 과정이다. 이것은 다른 조건은 동일하게 한 상태에서 아무런 일도 하지 않는 dummy MDA를 사용한 경우 수신 쪽에 bottle neck이 일어나지 않는 사실에 의해 확인할 수 있었다. 수신 쪽이 전자우편을 처리하다가 전자우편크기 증가와 같은 이유에 의해 MDA쪽 처리시간이 길어지면 전자우편을 받는 속도보다 처리하는 속도가 느려지므로 수신 쪽 기계 상에 queue를 처리하려는 process들의 수가 늘어나게 되어 load average가 증가한다. 그런데 전자우편크기가 커지면 식에 의해 msgpri의 값도 증가하므로 'msgpri > q / (la - x + 1)'을[4] 만족할 가능성도 커진다. 따라서 message크기가 증가할수록 sendmail이 deliver를 중단하고 queueing에 치중할 가능성이 커지는 것이다.

수신 쪽에서 전자우편을 처리하는데 MDA처리부분 다음으로 시간이 많이 걸리는 부분은 SMTP연결을 처리하는 부분이다[1]. 시스템의 부하에 따라 약간의 변동은 있지만 송신 쪽은 512 Byte의 경우 초당 27내지 28개의 SMTP연결을 처리하며 수신 쪽은 평균 초당 35내지 40개의 SMTP연결을 처리한다. 만약 이러한 숫자 이상의 SMTP연결 요구가 들어오는 경우 연결에 성공하지 못한 요구들은 약간의 delay를 두고 연결되지만 그 한계를 크게 넘는 SMTP연결 요구가 동시에 들어오는 경우는 수신 쪽 시스템이 견디지 못하고 오동작 하기도 한다.

4.3.2 시스템의 전자우편 수신 성능 측정 결과

---

4) 본 논문의 '2.1.2.3 sendmail의 동작 구조'부분에서 자세히 다루고 있다.

‘/etc/passwd'파일과 ’procmail'프로그램을 이용한 기존 시스템에서 1초 당 수신 가능한 전자우편의 수는 전자우편의 크기가 증가함에 따라 10통에서 4통으로 감소했다. 이 이상의 전자우편을 보내는 경우 본 논문의 관련 연구에서 언급한 ‘sendmail'의 동작 구조에 따라 ’sendmail'이 수신을 거부하는 결과를 볼 수 있었다.

사용자 인증과 메시지 요약정보 관리를 postgres DB로 하고 메시지마다 파일을 따로 두어 저장하는 시스템의 경우 전자우편의 크기에 거의 큰 영향을 받지 않고 초 당 0.9통의 전자우편을 수신하는 낮은 성능을 보였다. 이 경우 전자우편 한 통을 처리하는데 걸린 평균시간은 대략 1.07 ~ 1.09초 사이로 이러한 처리 시간의 96% ~ 99.9% 정도를 DB에 접근하는데 사용한다는 것이 특징이다.

본 논문에서 구현한 시스템의 경우 한 통의 전자우편을 보내서 수신하는데 걸리는 시간인 latency를 측정하여 이를 기준으로 1초 당 처리할 수 있는 최대 전자우편의 수를 추정하였다. 이렇게 한 이유는 앞에서 언급한 바와 같이 전자우편을 수신하는 경우 송신하는 측에서 초 당 송신하는 전자우편의 수를 증가시키면 디스크에 메시지를 쓰는 시간이 증가하여 전자우편 한 통을 수신하는데 걸리는 평균시간이 증가하는 경향을 보이므로 정확한 측정이 어렵기 때문인데 이러한 경향을 전자우편의 크기가 클수록 더 크게 나타난다. 실제로 초 당 25통의 전자우편을 수신하도록 한 경우 전자우편의 크기를 512 Byte, 4 KByte, 32 KByte, 256 KByte 로 변화시키면 한 통을 수신하는데 걸리는 시간과 latency와의 차이는 3ms, 5ms, 18ms, 100ms으로 커지는 것을 볼 수 있었다. latency를 통한 추정치는 초 당 30통에서 22통으로 그 격차가 적지만 초 당 25통의 전자우편을 수신하도록 한 경우의 측정치는 초 당 28통에서 7통으로 그 격차가 큼을 알 수 있다. 특히 256 KByte와 같이 크기가 큰 전자우편의 경우 그 변동폭이 심하고 초 당 3 통 이상의 비율로 전자우편이 도착하면 그러한 경향이 나타나므로 측정의 어려움이 있다. 이러한 문제로 인해 노드의 개수를 두 개로 한 경우의 실험은 256 KByte의 경우도 초 당 30통 이상의 전자우편을 받을 수 있다는 사실만을 확인하고 더 이상 진행하지 못했다.

그림6은 이러한 실험결과를 그래프로 나타낸 것으로 y축은 초 당 수신 통수를 나타내며 x축은 전자우편의 크기를 나타낸다. 표1은 그래프의 결과를 숫자로 나타낸 것이다.

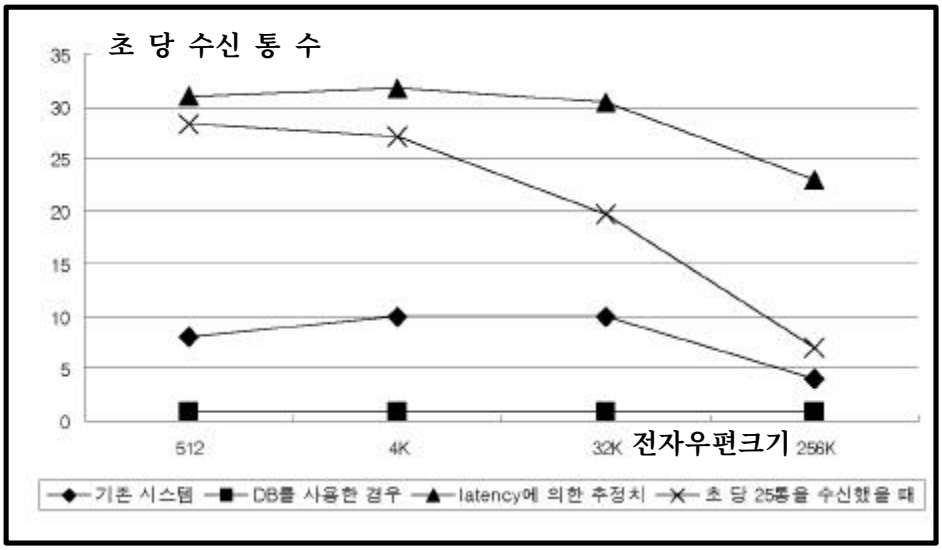

그림6 수신 성능 실험 결과

메시지크기	512 B	4 KB	32 KB	256 KB
기존 시스템	8	10	10	4
DB를 사용한 경우	0.929	0.931	0.928	0.911
latency에 의한 추정치	30.988	31.763	30.403	22.971
초 당 25통을 수신했을 때	28.261	27.167	19.695	7.033

표1 수신 성능 실험 결과표

5. 결론

'/etc/passwd'파일과 'procmail'을 이용한 기존 전자우편 서버시스템에서 수신 성능 저하의 원인이 되는 것은 MDA인 procmail의 속도가 느리다는 점과, 동시에 연결 가능한 SMTP connection수의 한계라는 사실을 실험을 통해 확인할 수 있었다. 본 논문에서는 이 같은 사실에 주목하여 전자우편 시스템에 대한 이해와 기존 연구들을 바탕으로 MDA로부터 사용자의 우편함까지 전자우편이 배달되는 부분을 수정한 웹-메일 서버 시스템의 프로토타입을 구현하였다. 이 과정에서 웹-메일 서비스를 제공하기 위해 사용한 데이터베이스가 성능 저하의 원인이 될 수 있다는 사실을 확인할 수 있었고, 사용자 각각에 대해 디렉토리를 만들어 메시지마다 하나의 파일로 저장하는 방식은 이 같은 데이터베이스 접근을 증가시키는 원인이 될 수 있다는 것도 알게 되었으므로 좀 더 나은 시스템으로 개선이 가능했다. 그 결과 MDA로부터 사용자의 우편함까지 전자우편이 배달되는 시간을 단축시켜서 단일 기계에 대한 단순 비교만으로도 기존 시스템에 비해 수신 성능을 약 두 배 가까이 향상시킬 수 있었다. 또한 리눅스 PC 클러스터를 이용하여 구성하여 동시에 연결 가능한 SMTP connection수가 증가한 효과를 볼 수 있을 것이므로 성능 확장의 가능성이 있는 웹-메일 서버 시스템을 가능케 할 수 있었다.

앞으로 round robin DNS대신에 virtual host와 같은 클러스터 구성 기법을 사용함으로써 성능 개선과 함께 load balancing과 같은 문제도 고려해야 할 것이다. 실험에서 나타난 '시간당 전송되는 전자우편의 수가 늘어날수록 디스크에 메시지를 쓰는 시간의 변동폭과 평균시간이 증가하는 현상'의 원인을 밝혀 성능을 개선하는 것도 해결해야 할 문제이다. 뿐만 아니라 성능 측정을 위한 보다 정확한 실험을 설계하고, 현재 구현하지 않은 가용성에 관한 부분을 추가하는 것도 필요할 것이다.

참고 문헌

[1] Michael Grubb - Duke University, How to Get There From Here: Scaling the Enterprise-Wide Mail Infrastructure, Proceedings of the Tenth USENIX Systems Administration Conference (LISA '96), Chicago, IL, 1996, pp. 131-138.

[2] Nick Christenson, Tim Bosserman, David Beckemeyer, EarthLink Network, Inc. A Highly Scalable Electronic Mail Service Using Open Systems, Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, California, December 1997

[3] Yasushi Saito, Brian Bershad, Henry Levy, and Eric Hoffman, The Porcupine Scalable Mail Server, Department of Computer Science and Engineering, University of Washington

[4] Yasushi Saito, Brian N. Bershad, and Henry M. Levy, Manageability, availability and performance in Porcupine: a highly scalable, cluster-based mail service, 17th ACM symposium on Operating System Principles (SOSP'99) Published as Operating Systems Review 34(5):1-15, Dec 1999

[5] Eric Allman, SENDMAIL - An Internetwork Mail Router, BSD UNIX Documentation Set, University of California, Berkeley, CA, 1986.

[6] David Wood, Programming Internet Email , Sebastopol, CA: O'REILLY & Associates, Inc., 1992

[7] Postel, Jonathan B. RFC 821: Simple Mail Transfer Protocol, 1982.

[8] Myers, J. and M.Rose. RFC 1725: Post Office Protocol - Version 3, 1994.

[9] Terry Gray, Message Access Paradigms and Protocols, http://www.imap.org/imap.vs.pop.html, 1995.08.28

[10] Terry Gray, Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP, http://www.imap.org/imap.vs.pop.brief.html, 1993.11.05

[11] Crispin, M. RFC 1730: Internet Message Access Protocol - Version 4, 1994.

[12] Sendmail, Home page, http://www.sendmail.org

[13] Sendmail, Home page, http://www.sendmail.net

[14] Bryan Costales with Eric Allman, sendmail ; Second Edition, O'REILLY, 1997.
[15] Cuccia, Nichlos H. "The Design and Implementation of a Multihub Electronic Mail Environment". San Die해, CA: USENIX Proceedings -- Lisa V; October 3, 1991.
[16] Albiz, Paul, and Cricket Liu. DNS and BIND. Sebastopol, CA: O'REILLY & Associates, Inc., 1992
[17] Thomas T. Kwan and Robert E. McGrath, NCSA's World Wide Web Server: Design and Performance, IEEE Computer, pp. 68-74, Nov. 1995.
[18] D. Dias, W. Kish, R. Mukherjee, and R. Tewari, A Scalable and Highly Available Server. COMPCON pp. 85-92, 1996.
[19] Wensong Zhang, Shiyao Jin, and Quanyuan Wu, Creating Linux Virtual Servers, In the proceedings of LinuxExpo, 1999.

Abstract

This paper addresses a web-mail server system using a cluster of Linux-based PCs.

The experiment shows that the low performance in general email server systems using '/etc/passwd' file and 'procmail' results from a slow MDA program and the limitation of concurrent SMTP connections.

In this paper, we modify the delivering procedure from MDA to user's mailbox in order to overcome such a problem. It is proved that the DB for a web-mail service and a wrong mailbox management approach can cause the low performance.

As a result, the implemented prototype system based on this observation, has about twice receiving performance as much as general mail server systems.

Because of an implemented web-mail system using linux PC cluster system, it has the advantage as if concurrent SMTP connections increases.

Keywords: email server system, cluster, mailbox, MDA, Linux PC

Student Number: 99419 - 542

감사의 글

먼저 이 논문을 쓰기까지 늘 자상하게 이끌어 주신 지도교수 하순회 교수님께 깊이 감사 드리며 논문 심사를 위해 시간을 내주시고 값진 조언을 해주신 전주식 교수님과 장래혁 교수님께도 감사를 드립니다.

웹-메일 서버의 구현에 많은 도움을 주신 홍익대학교 심영철 교수님께도 감사를 드리며 같이 일했던 강호석, 박은영씨에게도 이 기회를 빌어 감사의 마음을 전합니다.

항상 부족한 제가 포기하거나 게으름 피우지 않도록 도와준 양석이형에게 특히 감사드리고 지난 1년간 같이 고생하면서 큰 힘이 되어준 춘승이형에게도 감사드리고 싶습니다.

지금은 미국에 계시지만 항상 제 건강을 염려해 주시던 찬익이형, 여러 가지 질문을 해도 항상 친절하게 대답해주는 현옥이형, 여러 가지 새로운 기술과 아이디어에 대한 이야기로 연구에 도움을 준 도형이형, 같이 수업을 들으며 많은 도움을 준 채석이형, 이것저것 조언을 많이 해주는 성찬이형, 연구실 생활에 적응하도록 많은 도움을 준 현욱이형, 이제는 아이 아버지가 되신 자상한 강녕이형, 항상 열심히 연구하는 모범을 보여준 민영이, 많은 도움을 주지 못해서 미안한 영민이, 마음씨 착한 채은이, 이 모든 연구실 사람들에게도 감사 드립니다. 또 늦게 까지 논문을 쓰는 선배를 걱정해준 신육이와 희석이에게도 고맙다는 말을 하고 싶습니다.

끝으로 늦게까지 논문을 쓰는 자식 때문에 걱정이 많으셨던 부모님께 감사드립니다.