효율적이고 신뢰성 있는 네임 서비스 시스템의 구조 및 실험

정회원 심 영 철, 박 준 철, 강 호 석, 이 준 원

An Architecture and Experimentation for Efficient and Reliable Name Service Systems

Young-Chul Shim, Jun-Cheol Park, Ho-Seok Kang, Jun-Won Lee *Regular Members*

요 약

네임 서비스 시스템은 도메인 네임을 인터넷 주소로 또는 역으로 인터넷 주소를 도메인 네임으로 변화하는 서비스를 제공하며 웹, 이메일, 파일전송 등의 모든 인터넷 응용이 가장 먼저 사용하는 인터넷에서 가장 중요한 서비스 중의 하나이다.

본 논문에서는 상위 레벨의 대규모 도메인을 가정하여, 이 도메인에 대한 서비스를 제공하는 네임서버를 중심으로 성능 향상 및 신뢰성 제고의 방안을 제시하고, 테스트베드 구축 및 성능 실험을 통해 이의 실현 가능성을 검증한다. 제안한 구조는 이중화된 네임서버를 가정하여 부하 분담 및 고신뢰성 달성이라는 두 가지 목표를 동시에 이루고자 하였다. 또한 매스터 네임서버의 데이터 갱신을 위해 동적 갱신 방식을 사용하였고, 슬레이브 네임서버로의 데이터 전송을 위해 IXFR(Incremental Zone Transfer)을 사용하여 서버의 부담을 최소화하고자 하였다. 테스트베드의 실험 결과를 실제 구축될 시스템에 적용할 경우, 향후 서버에의 부하 증가에 효율적으로 대처할 수 있으며 시스템의 다운이나 오동작의 경우에도 연속적인 서비스 수행이 가능하다고 판단된다.

ABSTRACT

Domain Name System(DNS), one of the most important Internet services, handles mapping from host names to Internet addresses and vice versa, and precedes many Internet applications such as Web, e-mail, file transfer, etc.

In this paper, we propose a structural design of a generic name server system providing name services for a huge domain for the purpose of improving the performance as well as the reliability of the system. We demonstrate the validity of the design by implementing and running a testbed system. Our testbed employs a couple of master name servers for distributing the service overhead over two, rather than one, servers and for achieving high availability of the system as a whole. We suggest the use of dynamic update to add and delete records from a zone for which the name server has authority. The slave name servers located remotely then get a new, updated copy of the zone via incremental zone transfers(IXFRs). The experiments with the implemented testbed show that the proposed structure would easily manage increasing demands on the server power, and be highly available in the face of transient faults of a module in the system.

Ⅰ. 서 론

네임 서비스[1,2]는 도메인 네임을 인터넷 주소로 또는 역으로 인터넷 주소를 도메인 네임으로 변화하는 서비스를 제공하며 웹, 이메일, 파일전송 등의 모든 인터넷 응용이 가장 먼저 사용하는 인터넷에서 가장 중요한 서비스 중의 하나이다[3]. 그러므

* {shim,jcpark,hskang,junlee}@cs.hongik.ac.kr
논문번호 : 030275, 접수일자: 2003년 6월 27일

로 네임 서비스 시스템의 효율적이고 안정적인 운용은 인터넷 서비스 제공에 있어서 매우 중요하다고 볼 수 있다[4]. 네임 서비스 시스템에서 가장 중요한 역할을 하는 것은 네임서버이며[5], 본 논문에서는 이러한 서버의 수용 능력을 향상시켜 시스템 과부하를 방지하고, 신속하고 견고한 갱신 체계를 구축하여 최신의 데이터로 서비스를 제공하며, 시스템 오동작이나 다운에 대비하여 신뢰성을 높이는 것들에 대한 방안을 제시하고자 한다. 네임 서비스를 하기 위한 프로그램으로 가장 대표적인 것이 BIND가 있다. BIND의 최신 버전은 9.2.1[6]이고, 본 논문에서는 BIND를 사용하여 DNS 서버를 구성하였다[7].

본 논문에서는 임의의 대규모 상위 도메인에 대한 네임 서비스를 제공하는 네임 서버를 중심으로 성능 향상 및 신뢰성 제고의 방안을 제시하고, 테스트베드 구축 및 성능 실험을 통해 이의 실현 가능성을 보인다. 제시된 방안을 따르는 구조는 성능, 신뢰성, 간단한 기본 구조에서의 신속한 전이 등의 측면에서 가장 적절하다고 판단한 시스템 구조이다. 이 구조는 또한 추후 필요에 따라 성능이나 신뢰성 측면의 더욱 강화된 요구 사항이 발생하는 경우 점차적인 확장이 용이하도록 구성되었다. 본 논문에서는 가장 단순하고 기본적인 네임 서버 운용 환경에 대한 분석에서 출발하여, 향후 시스템 수용 능력을 확장하기 위한 방편으로 이중화된 네임 서버 운용을 제안한다. 그리고 이러한 구성에 기반하여 네임 서비스 제공을 위한 최신의 파일을 유지하고, 이를 다수의 외부 네임 서버들에 반영하는 기법을 제시하며 이의 타당성 및 성능 관련 측정 내용을 논증한다.

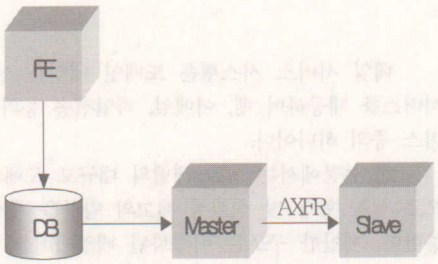
본 논문의 구성은 다음과 같다. 2장에서는 단순한 기본 구조에서 출발하여 다양한 구조를 제시 분석함으로써, 최소한의 전이를 통해 달성할 수 있는 성능 제고를 위한 구조를 제안한다. 3장에서는 고성능과 고신뢰성을 가지며 저비용에 구축될 수 있는 네임서비스 시스템의 구조를 제안한다. 4장에서는 4장의 제안 구조를 반영한 테스트베드의 구현과 실험 결과를 서술한다. 5장에서는 제안한 구조에서 고신뢰성을 위한 모듈간 프로토콜을 제시한다. 6장에서는 연구의 의미와 결과를 요약하고 결론을 맺는다.

Ⅱ. 고성능 네임 서비스 시스템 구조

본 장에서 가장 기본적인 네임 서비스 시스템 구성에서 시작하여 성능을 향상시킬 수 있는 다양한 구조를 제시하고 이들을 비교 분석한다.

2.1 기본적 네임 서비스 시스템의 구조

가장 기본적인 형태의 네임 서비스 시스템의 구성은 (그림 2.1)과 같다.



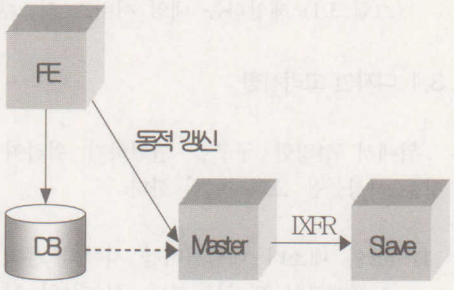
(그림 2.1) 기본적 네임 서비스 시스템의 개념적 구성

외부 도메인 네임 등록기관(Registrar)으로부터 전달된 도메인 네임의 추가, 삭제, 변경 등 갱신 요구사항은 FE(Front End)에 전달되고 이 도메인 정보의 변경 요구들은 DB 서버에 저장된다. 정해진 주기(예를 들어, 8시간마다)에 DB 서버의 도메인 정보를 컴파일하여 존 파일을 만들고 이를 매스터 네임서버에 전달하여 매스터 네임서버가 새로운 존 파일을 가지고 동작하도록 한다. 그리고 매스터 네임서버가 슬레이브 네임서버에게 변경된 존 파일의 전부를 트랜스퍼 한다(AXFR(All Zone Transfer))[8]. 이 구조에서의 문제점은 관장하는 도메인의 크기가 큰 경우, 먼저 DB 서버 내의 도메인 정보를 전부 컴파일하여 전체 존 파일을 새로 만든 후 이를 매스터 네임서버에게 보내는 전체 과정이 시간이 많이 걸리므로 이를 자주 수행하지 못하고 하루에 세 번이나 네 번 정도 수행되는 것이 보통이다. 결과적으로 사용자의 도메인 정보 갱신 요청이 최악의 경우 한 주기(6시간이나 8시간)의 시간 후 반영되는 경우가 발생할 수 있다. 두 번째로 매스터 네임서버에서 슬레이브 네임서버로 전체 존

파일을 전송하므로 네임서버들에 걸리는 부하도 크고 존 파일의 전송에도 시간이 많이 걸린다는 문제가 있다.

2.2 동적 갱신(Dynamic Update)과 IXFR (Incremental Zone Transfer)을 이용한 구조

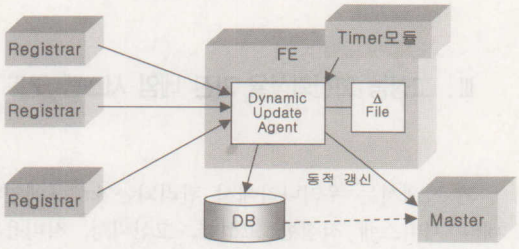
앞 절에서 소개한 기본적인 네임 서비스 시스템의 성능을 향상시키기 위하여 (그림 2.2)와 같이 동적 갱신[9]과 IXFR[10]을 사용할 수 있다. FE를 통해 들어온 사용자 요청은 DB 서버로 전달이 되어 저장되고 동시에 FE에서는 이 변경 요청이 델타 파일 내에 저장된다. 델타 파일의 크기가 충분히 커지거나 또는 충분한 시간이 지난 후, FE는 델타 파일 내의 내용을 사용하여 DNS 동적 갱신 패킷을 만들고 이를 매스터 네임서버에게 보낸다. 이를 통해 축적된 변경 요구 사항이 모두 매스터 네임서버에 반영되도록 한다. 이 과정이 끝나면 FE에서의 델타 파일 내의 내용은 모두 삭제된다. 그리고 매스터 네임서버는 동적 갱신에 의해 변경된 내용을 IXFR을 사용하여 외부 슬레이브 네임서버들에게 전달한다. 이 구조에서는 매스터 네임서버의 존 파일을 갱신하기 위하여 동적 갱신을 사용하므로 DB 서버의 내용으로 전체 존 파일을 만들 필요가 없다. 따라서 매스터 네임서버에의 갱신을 위한 오버헤드가 줄어들게 되어 현재의 방법보다 자주 네임서버의 내용을 갱신할 수 있게 된다. 그리고 슬레이브 네임서버로의 존 트랜스퍼도 IXFR을 사용하게 되므로 역시 매스터 네임서버에서의 오버헤드가 많이 감소하게 된다. 데이터베이스의 내용은 매스터 네임서버의 존 파일에 문제가 발생하여 존 파일을 처음부터 다시 만들어야 하는 경우 사용된다.



(그림 2.2) 동적 갱신과 IXFR을 사용한 네임 서비스 환경

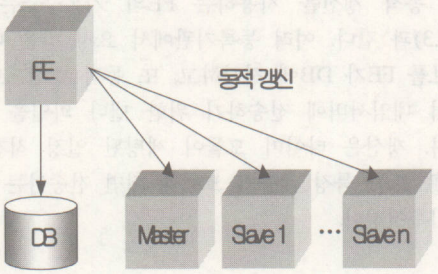
동적 갱신을 사용하는 FE의 기본구조는 (그림 2.3)과 같다. 여러 등록기관에서 요청 받은 네임 정보를 FE가 DB에 저장하고, 또 동적 갱신으로 매스터 네임서버에 전송하기 위한 델타 파일을 보관한다. 갱신은 타이머 모듈이 세팅된 일정 시간 간격 후, 또는 특정 개수가 모이게 되면 전송하는 식으로 이루어진다.

(그림 2.3) FE의 구조



2.3 동적 갱신만을 사용한 구조

(그림 2.2)의 구조에서 매스터 네임서버가 여러 슬레이브 네임서버들에게 존 트랜스퍼를 하여야 한다는 것은 매스터 네임서버에게 성능상의 부담이 될 수 있다. 이를 해결할 수 있는 방법 중의 하나가 (그림 2.4)와 같이 동적 갱신만을 사용하는 방법이다. 이 구조에서는 FE에 축적된 사용자 변경 요구 사항들을 매스터 및 모든 슬레이브 네임서버에게 동적 갱신에 의해 전달한다. 이러한 경우 슬레이브 네임서버들은 이름만 슬레이브이지 실제로는 매스터 네임서버의 형태로 구성이 되어야 동적 갱신이 가능하다. 이러한 구조에서는 매스터 네임 서버에서의 존 트랜스퍼로 인해 발생하게 되는 오버헤드가 완전히 제거되게 된다. 그러나 매스터 네임서버 또는 슬레이브 네임서버들이 다운되어 있는 동안 전달된 동적 갱신의 내용이 어떻게 궁극적으로, 그리고 올바르게 반영될 수 있는가의 문제가 해결되어야 한다.



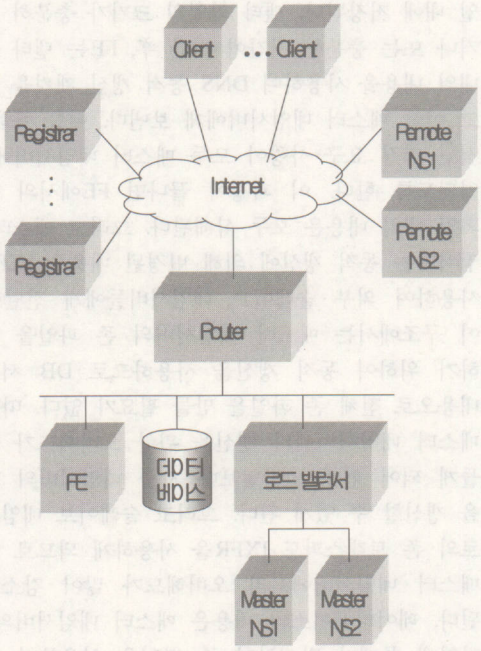
(그림 2.4) 동적 갱신만을 사용한 네임 서비스 환경

Ⅲ. 고성능 고신뢰성을 가진 네임 시스템 구조

본 장에서는 우리나라에서 관리되는 kr 도메인의 네임 서비스에 적절한 고성능, 고신뢰성, 저비용의 네임 서비스 시스템의 구조를 설명하고자 한다. 성능을 향상시키기 위해서는 2.3절에서 설명한 동적 갱신만을 이용한 구조가 적절할 것이나 이 구조에서는 일부의 네임 서버들에 문제가 발생하는 상황에서도 궁극적으로 모든 네임 서버들이 같은 존 파일을 가지게 하는 것이 기술적으로 복잡하다. 따라서 약간의 성능 저하는 있을 수 있으나 훨씬 안정적으로 운용될 수 있는 2.2절의 구조를 사용하고자 한다. 현재 kr 도메인에서의 네임 서비스 요청은 대부분 매스터 네임 서버로 전달되므로 매스터 서버의 성능과 가용성을 증가시키는 것이 중요하다. 이를 위하여 이중화된 두개의 매스터 네임 서버를 채용한다. 반면 지역적으로 분산 배치되어 있는 슬레이브 서버들은 사용 빈도가 매스터 서버에 비해 낮으므로 단일서버를 채용하여 설치비용이 과다하게 상승하지 않도록 한다.

(그림 3.1)에서 이러한 구조를 보여주고 있다. 등록기관의 도메인 네임 갱신 요청은 FE에게 전달되고 이는 즉시 데이터베이스에 저장된다. 네임 갱신 요청은 동적 갱신을 통하여 두 개의 매스터 네임 서버에 반영되고 IXFR을 통하여 슬레이브 서버에까지 전달된다. 매스터 서버에 전달되는 클라이언트의 네임 서비스 요청은 로드 밸런서(Load Balancer)에 의하여 두 개의 매스터 서버에 균등하게 분배된다. 만약 한 개의 매스터 서버가 죽게 되면 다른 매스터 서버가 모든 네임 서비스 요청을 처리하게 된다.

각 클라이언트들은 매스터 네임서버의 논리주소(두개의 매스터 네임 서버가 공유하는 IP 주소)를 이용하여 네임 서비스를 받는다. 매스터 네임서버들의 물리주소(각 네임서버의 실제 IP주소)를 이용하지 않고 논리주소를 사용함으로써 네임서버의 부하분산을 로드 밸런서에게 완전히 일임할 수가 있고 특정 네임서버의 고장 또는 교체로 인한 서비스가 중단될 경우에 클라이언트들은 이러한 것에 상관없이 논리주소로 접근하여 지속적으로 서비스를 받을 수 있는 장점이 있다. 그러나 FE와 원격 네임서버는 동적 갱신 및 IXFR을 위하여 네임서버의 물리주소를 이용한다.



(그림 3.1) 제안하는 네임 서비스 시스템 구조

3.1 디자인 고려사항

위에서 설명한 구조를 실현하기 위하여 다음과 같은 사항들을 고려하여야 한다.

1) FE와 매스터 네임서버들 사이의 동적 갱신은 트랜잭션화 되어야 한다. 성능상의 이유로 FE에 들어온 갱신 요청은 즉시 매스터 네임서버에 동적 갱신 요청으로 전달되지 않고 어느 정도 요청들이 쌓인 후 또는 정해진 시간이

지난 후 한꺼번에 매스터 네임서버에 전달하는 것이 일반적이다. 매스터 네임서버는 전달된 동적 갱신 요청을 트랜잭션으로 처리한다. 즉 네임서버의 메모리 자료구조를 갱신함은 물론 갱신 내용을 비휘발성 저장 장치에 로그 파일로 남긴다. 그리고 이후 FE에게 동적 갱신 요청이 성공적으로 수행되었다는 응답 코드를 보낸다. 만약 성공 응답 코드를 정해진 시간 내에 받지 못하면 FE는 동적 갱신을 다시 수행하여야 한다. 만약 실패 응답 코드를 받으면 동적 갱신 요청이 실패한 이유를 분석한 후 수정된 동적 갱신 요청을 다시 보내야 한다.

2) SOA(Start of Authority) 자원 레코드의 SERIAL 갱신에 주의하여야 한다. 동적 갱신 표준 문서에 의하면 SOA SERIAL은 ① 매 갱신 후, ② 최종 갱신을 수행하고 정해진 시간이 경과한 후, 또는 ③ 정해진 수의 갱신이 수행된 후 변경될 수 있다[9]. 더욱 문제를 복잡하게 만드는 것은 매스터 네임서버의 SOA SERIAL이 변경된다 하더라도 이 내용이 즉시 존 파일에 반영되지 않을 수 있고 따라서 언제 SOA SERIAL이 변경되었는지 알 수 없다. 존 파일의 SOA SERIAL은 네임 서버에 의해 결국 갱신이 되겠지만 DB 내의 SOA SERIAL의 변경을 어떻게 매스터 네임서버의 SOA SERIAL 변경과 일치시킬 수 있는가가 문제가 될 수 있다. 이것이 가능하기 위하여서는 매스터 네임서버가 SOA SERIAL을 언제 어떠한 값으로 변경하는지 그 내용을 FE가 알아서 이에 따라 DB의 내용을 변경하는 것이 필요하다. 현재 우리나라를 포함 전 세계적으로 가장 널리 사용되는 DNS 소프트웨어인 BIND (Berkeley Internet Name Domain)를 기준으로 보면, BIND 버전 8에서는 5분 또는 갱신을 위한 엔트리가 100개가 되었을 때 (둘 중 먼저 오는 것이 우선) SOA SERIAL을 증가시키고, BIND 버전 9의 경우 네임서버가 동적 갱신을 진행할 때마다 SOA SERIAL을 1씩 증가시켜 로그 파일(.jnl)에 저장한다.

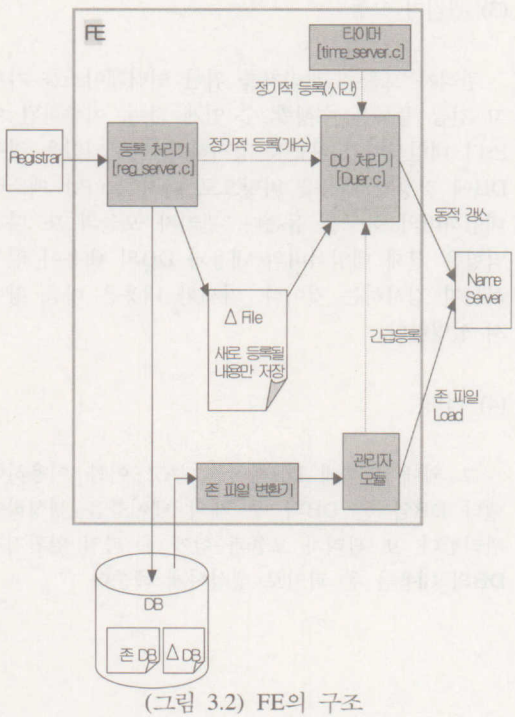
3) 슬레이브 네임서버로의 존 트랜스퍼를 언제 하느냐가 문제가 될 수 있다. 매스터 네임서버의 내용과 슬레이브 네임서버의 내용 사이의 일치성(consistency)을 보장하기 위하여 매스터 네임서버에서 동적 갱신이 수행되자마자 SOA SERIAL을 증가시키고 이를 NOTIFY 메시지 [11]로 슬레이브 네임서버로 알린 후, 이에 따라 슬레이브 네임서버가 존 트랜스퍼를 요청하여야 한다.

3.2 FE의 구조

매스터 네임서버에의 동적 갱신을 위주로 본 FE의 전반적인 구조는 (그림 3.2)와 같다. 각 모듈의 기능 및 동작 방식을 간략히 서술한다.

(1) 등록 처리기

등록 처리기의 주된 기능은 등록기관으로부터 DNS 데이터의 갱신을 요구받으면 이 데이터의 형식을 검사하고 델타 DB, 존 DB, 델타 파일에 기록하는 것이다. 델타 DB, 존 DB, 델타 파일을 구별하여 운용하는 이유는 네임서버의 갑작스런 고장에 대처하기 위한 것이고 그 내용은 5장에서 설명한다. 여러 등록기관에서 등록 처리기를 호출한 경우 DB와 새로 등록될 내용을 가진 파일에 lock을 걸고 쓰는 작업을 수행한다.



(그림 3.2) FE의 구조

(2) 동적갱신(DU) 처리기

DU 처리기는 등록 처리기에서 생성한 파일을 이용하여 네임 서버에 동적 갱신을 수행한다. 시간 기반에 의한 등록, 개수에 의한 등록, 긴급 등록의 세 가지가 가능하다. 시간 기반 등록은 관리자가 설정한 타이머에 의하여 일정 시간마다 DU 처리기가 호출되어 각 네임서버에 동적 갱신을 수행함을 의미한다. 개수에 의한 등록은 등록 처리기가 일정 건수(예, 1,000개)에 도달하면 DU 처리기를 호출하여 동적 갱신을 수행하는 것을 의미한다. 마지막으로 관리자가 긴급 등록을 요구할 경우 지금 막 요청한 것을 포함하여 지금까지 저장된 모든 요청 내용을 DU 처리기를 통해 등록할 수 있다. 이 때 누가, 언제, 무슨 내용 때문에 긴급등록을 했는지 로그를 남긴다(또는 시스템 관리자의 허락 하에 긴급 등록을 수행). 시간 등록, 건수 등록, 긴급 등록이 겹쳐서 수행되거나 관리자 모듈에서 긴급 등록을 동시에 할 경우 문제가 발생하므로 이를 해결하기 위해, 새로 등록될 내용을 포함하고 있는 파일에 lock을 걸어서 단지 한 개의 프로세스만 파일의 내용을 가져가게 되고 파일 내용을 가져가면 반드시 파일 내용을 없애도록 한다.

(3) 관리자 모듈

관리자 모듈은 관리자를 위한 인터페이스를 가지고 긴급 등록을 수행할 수 있게 하며, 이중화된 매스터 네임서버의 다운을 감시하여, 다운되었을 경우 DB에 저장된 내용을 바탕으로 다시 살아난 매스터 네임서버의 복구를 돕는다. 관리자 모듈의 또 다른 역할은 실제 네임서버의 내용과 DB의 내용이 일치하는지 검사하는 것이다. 자세한 내용은 다음 장에서 설명한다.

(4) 기타

그 외에 외부에 DB서버를 두고 이를 이용하여 델타 DB와 존 DB의 두 개의 테이블을 생성하여 관리한다. 또 관리자 모듈과 함께 존 파일 변환기가 DB의 내용을 존 파일로 생성하게 해준다.

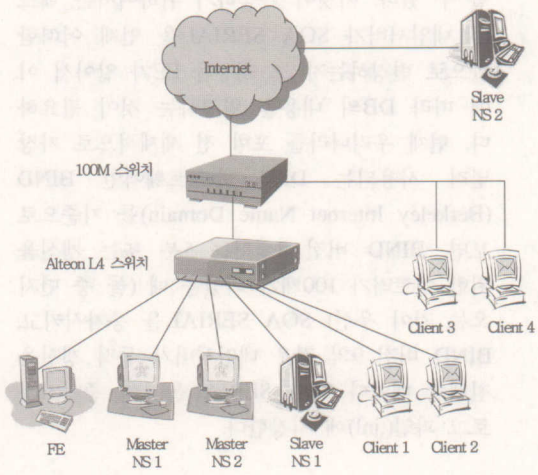
Ⅳ. 테스트베드 구현과 실험 결과

본 장에서는 제안한 구조의 실용성을 검증하기 위한 테스트베드의 구축과 성능 관련 실험 결과들을 제시한다.

4.1 테스트베드의 구성

이중화된 매스터 네임서버에의 동적 갱신과 슬레이브 네임서버에의 IXFR에 기초한 테스트베드를 (그림 4.1)과 같이 구성하였다. 부하 분산을 위해 Alteon L4 스위치[12]를 네임서버 상단에 위치시키며, FE, 매스터 네임서버 및 슬레이브 네임서버, 그리고 테스트를 위한 클라이언트들은 본 기관 내부에, 또 다른 슬레이브 네임서버 하나는 본 기관 외부에 설치하였다.

전체적으로 3장에서 제안한 구조로 테스트베드를 구성하였으나 구현의 편의성을 위해 라우터와 로드밸런서의 역할을 하나의 Alteon L4 스위치가 모두 담당하도록 하였고, Alteon L4 스위치 상에 다른 스위치를 하나 따로 두어서 외부 클라이언트 역할을 하는 컴퓨터를 연결하였다. 하지만 성능 및 신뢰성을 위한 프로토콜의 측면에서 제안한 구조의 특성이 그대로 나타날 수 있도록 하였다.



(그림 4.1) 이중화된 매스터 서버에 동적 갱신 및 IXFR을 적용한 구성

네임서버는 BIND 8.3.4[13]를 운용한다. 현재 BIND의 버전은 9.2.1까지 나와 있으나, 현재 안정적으로 다수의 시스템에서 사용하고 있는 BIND 8.3.4를 테스트베드에서도 사용하였다. 하지만 nslookup[14]과 nsupdate 함수를 이용하기 위해서는 BIND-utils를 설치하여야 하는데 BIND-utils 버전 8.3.4이 제대로 동작하지 않기 때문에 BIND-utils는 버전 9.2.1을 설치하여 테스트하였다. BIND 버전 9와 BIND 버전 8의 가장 큰 차이점은 동적 갱신 및 IXFR을 위한 로그파일의 운용 방식이다. BIND 버전 9의 경우 동적 갱신과 IXFR을 위하여 하나의 로그 파일(.jnl)만을 생성하고 삭제되지 않으며 지속적으로 관리되는 반면에, BIND 버전 8의 경우 동적 갱신을 위한 로그 파일(.log)과 IXFR을 위한 로그 파일(.ixfr) 두 개를 유지한다. 그리고 동적 갱신한 데이터를 존 파일에 쓰거나 IXFR을 모두 마쳤을 경우에 로그 파일들은 삭제된다. 또 다른 차이점은 BIND 버전 9의 로그 파일의 경우 관리자가 읽을 수 없는 binary 형태로 되어 있으나, BIND 버전 8의 경우 로그 파일을 관리자가 읽을 수 있는 형태로 되어 있다[3]. 동적 갱신된 데이터들이 존 파일에 쓰여질 경우 SOA SERIAL은 BIND 버전 9와 BIND 버전 8 모두 nsupdate 트랜잭션당 1씩 증가하는 것을 확인하였다.

Alteon L4 스위치는 8개의 10/100 Mbps 포트, 1개의 1000BASE-SX 포트, 포트당 2MB 메모리, 336K 동시 세션 처리 등의 특징을 가지고 있고, 8 Gbps의 스위치 Backplane을 지원한다. IP 트래픽의 제어 및 보안을 위해 포트당 224개의 필터(Packet Rule)를 제공하며 서버 로드 밸런싱, firewall 로드 밸런싱, 응용 redirection 등 여러 가지 환경에 맞추어 다양한 로드 밸런싱을 할 수 있다[14]. 이 스위치에 가상 IP(3장에서의 논리주소)를 세팅하여 클라이언트들은 가상 IP를 매스터 네임 서버에 대한 질의를 하도록 하였다. 스위치가 클라이언트로부터 DNS 질의를 받으면 부하 균등화 메커니즘에 의해 두 개의 매스터 네임서버 중에 하나로 질의를 전달한다. 그 매스터 네임서버로부터 응답이 오면 그 응답을 자신의 가상 IP를 이용하여 클라이언트에게 전달함으로써 두 개의 매스터 네임서버의 부하분산을 하였다. 또한 하나의 매스터 네임서버가 동작하지 않을 경우 스위치는 이를 탐지하여 현재 동작중인 네임서버에게만 질의를 보내어 서비스의 중단을 방지할 수 있으며, 실제 네임서버의 IP가 공개되지 않으므로 보안의 측면에서 유리하다.

슬레이브 네임서버는 존 파일을 전송 받기 위한 주 매스터 네임서버를 설정하여야 한다. 테스트베드에서 슬레이브 네임서버의 주 매스터 네임서버로서 이중화된 두 개의 매스터 네임서버 모두를 설정하여 사용한다. 하나의 매스터 네임서버만을 주 매스터 네임서버로 설정하면 부하 분산 측면에서 효율적이지 못하며, 또한 설정한 매스터 네임서버가 동작을 하지 않을 때 슬레이브 네임서버는 더 이상 갱신된 새로운 데이터를 받을 수 없게 된다. 따라서 두 개의 매스터 네임서버를 모두 슬레이브 네임서버의 주 매스터로 설정해 놓고, 하나의 네임서버가 동작하지 않을 경우에 다른 쪽 네임서버에서 IXFR을 수행할 수 있도록 하였다.

4.2 테스트베드 상의 성능 측정

실험의 목적은 제안된 구조를 실험적으로 검증하고자 하는 것으로서, 실제 네임 서버를 운용할 환경에서 제안된 구조를 따르는 시스템이 얼마나 잘 동작할 것인지 입증하는 근거를 제시하고자 한다. 우선 4.2.1에서 이전 구조에 비해 가장 큰 변동 사항인 동적 갱신에 대해 그 안전성과 성능 측면의 부하 정도를 측정한다. 이를 위해 운용 중인 테스트베드 환경에서 대규모의 동적 갱신을 적용시켜 응답시간을 측정함으로써 동적 갱신 자체의 안정성과 시스템에의 영향을 관측한다. 실험한 데이터 규모는 초기의 기존 데이터를 새로운 서버 시스템에 적용하는 과정에서 요구되는 양 이상으로 볼 수 있어 동적 갱신의 영향력 판단에 객관적인 결과를 제공한다. 다음으로 안정적으로 운용되는 테스트베드에 실제 동적 갱신의 비율을 고려하여 전체 데이터의 10% 및 5%의 동적 갱신 요구를 적용시키고 응답시간을 측정하였다. 이를 통해 운용중인 시스템에서 실제 발생 가능한 상황을 가장 잘 모사하여 동적 갱신의 영향을 살펴본다. 이제 4.2.2에서는 제안한 이중화 구조가 한 서버의 다운시 어떻게 가용성을 계속 보장하는가를 성능 측면에서 살펴본다. 예상했던 것처럼 실험의 결과 한 서버가 다운되더라도 그 서버의 재시동 시점까지 살아있는 나머지 서버로 네임 서비스를 계속해서 제공하는 것이 가능함을 보여준다. 이 결과는 단위 서버의 서비스 처리량이 전체 전체 서비스 요구량보다 적지 않다는 가정 하에서 성능 측면에서도 서버 다운 시점을 제외하고

<표 4.1> 실험에 사용한 시스템 사양

	느린 컴퓨터 (매스터 서버)	빠른 컴퓨터 (매스터 서버)	Alteon 스위치 사용시 매스터서버	슬레이브 서버
CPU	600MHz	1.7GHz	600MHz X 2	600MHz
RAM	128M	256M	128M X 2	128M
HDD	20GB	60GB	20GB X 2	20GB
	FE	클라이언트 1,3	클라언트 2	클라이언트 4
CPU	600MHz	1.0GHz	1.7GMHz	466MHz
RAM	128M	256M	256M	128M
HDD	20GB	40GB	60GB	10GB

는 전체 응답시간 지연이 확연히 늘지 않았음을 보여준다. 마지막으로 4.2.3에서 동적 갱신의 두 가지 연산 ADD와 DELETE 각각이 성능에 미치는 영향을 측정한다. 이 실험을 통해 서버 컴퓨터와 존 파일의 크기에 따라 수용할 수 있는 각 연산의 수가 결정되기 때문에, 실제 시스템 운용시 동적 갱신의 빈도를 갱신 개수에 따라 결정할 필요가 있음을 확인한다.

실험은 Alteon L4 스위치를 이용하여 이중화된 두 대의 매스터 네임서버에 적절히 부하를 분산시키는 방법으로 진행하였다. 아래 <표 4.1>은 실험에 사용한 시스템 사양이다.

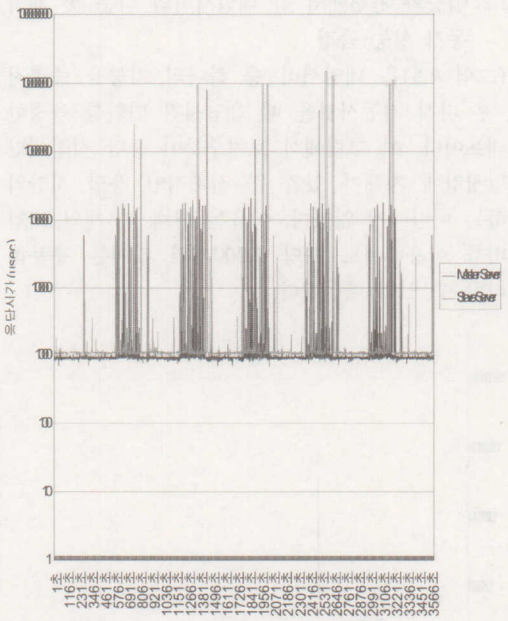
실험에 사용한 BIND의 버전은 8.3.4이고, 동적 갱신에 사용한 프로그램은 nsupdate이다. 클라이언트에서 응답시간을 측정하기 위한 질의는 gethostbyname()함수[15]를 변형하여 사용하였다. 실험은 네임서버에 초당 1,000개의 질의가 항상 들어온다고 가정을 하고 측정을 하였다. 이 초당 1,000개의 질의는 클라이언트 1과 2가 각각 초당 500개의 질의씩 분담하여 수행한다. 질의에 대한 응답 시간을 측정하는 것은 클라이언트 3과 4에서 수행하였다. 클라이언트 3은 매스터 네임서버에 대한 응답시간을 측정하고, 클라이언트 4는 슬레이브 네임서버에 대한 응답시간을 측정하였다.

4.2.1 동적 갱신 관련 성능 측정

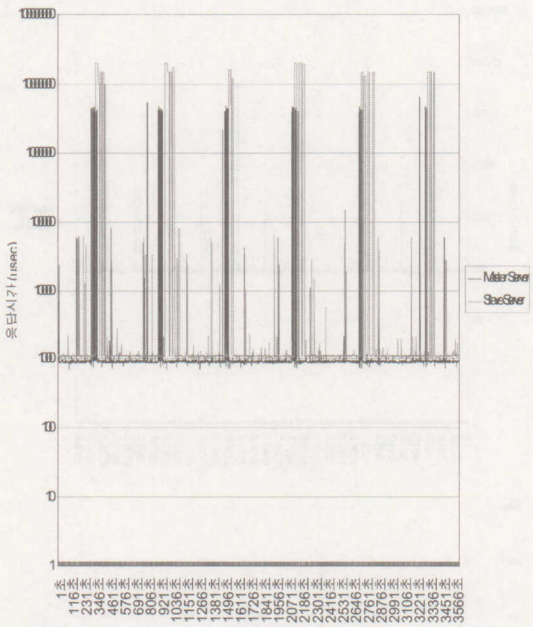
부하 분산은 Alteon L4 스위치를 이용하여 600MHz 두 개의 서버를 라운드 로빈 방식으로 동작하도록 하였고, 동적 갱신은 두 개의 네임서버에 순차적으로 보냈다. 즉 네임서버1의 동적 갱신이 완료되면, 다음 네임서버2에 동적 갱신을 수행한다.

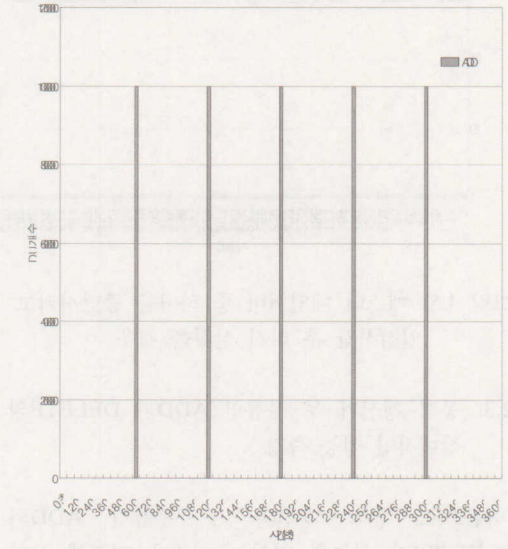
(그림 4.2)는 초기화 상태에서 10분 간격으로 동적 갱신을 1부터 500,000개까지 넣은 값이고, (그림 4.3) 및 (그림 4.4)은 전체 존 파일 데이터의 크기를 500,000개로 설정하고 하루에 변화되는 양이 각각 전체 크기의 10% 및 5% 일 때 이에 맞게 한 시간 동안 실험을 한 내용이다.

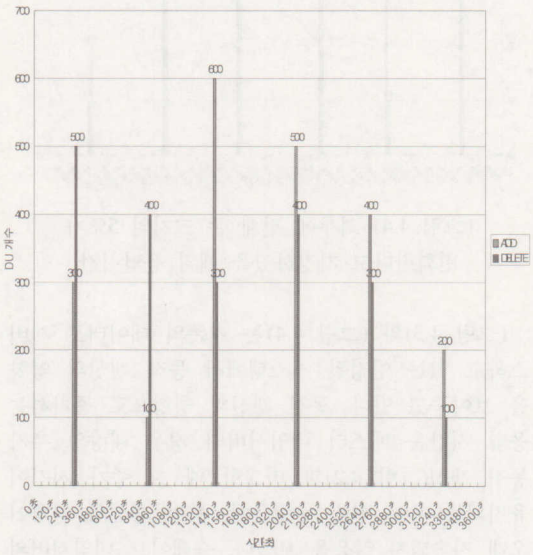
이 실험을 통해 100,000개의 데이터를 추가하는 작업은 10 분 주기의 약 1/2의 정도까지 응답 시간에 영향을 미침을 알 수 있다. 응답 시간은 동적 갱신의 영향이 미치지 않는 경우 1,000 us(1 ms) 정도이며, 동적 갱신이 수행되는 도중에는 약 100 배(100,000 us)에서 최대 슬레이브 네임서버의 경우 10,000 배(10,000,000 us)까지도 증가하는 모양을 나타냈다. 동적 갱신 추가의 경우 상당히 큰 규모인 100,000 개의 데이터 갱신은 시스템의 초기화 상태에서 기존 데이터의 업로드 시 일어나는 전체 데이터 갱신의 규모와 비교할 수 있을 정도로 큰 작업이다. 이로부터 초기의 기존 데이터를 새로운 서버 시스템에 적용하는 과정은 서비스 요구가 가장 적은 시점을 택하여 신속하게 수행할 필요가 있다고 판단된다.



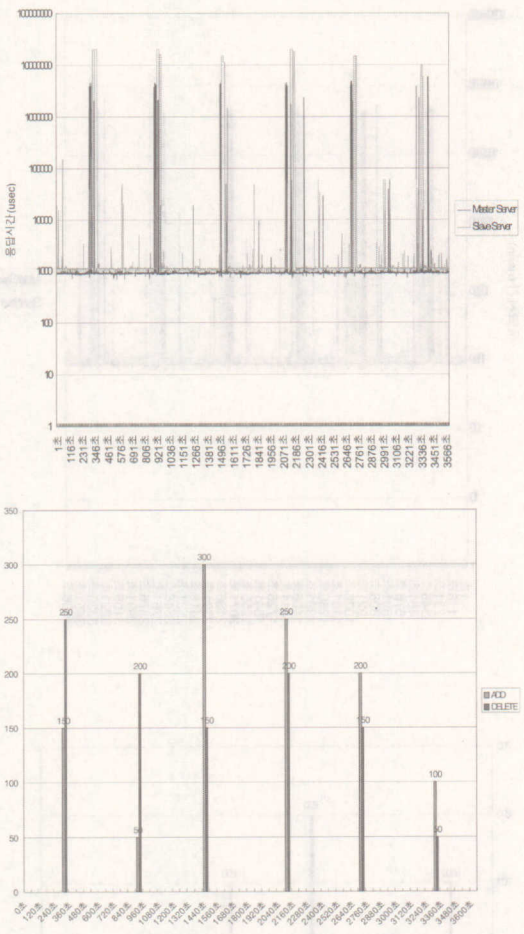
(그림 4.2) 동적 갱신을 1부터 500,000 까지 증가시킬 때의 응답시간







(그림 4.3) 하루에 전체 존 크기의 10%가 변화한다고 가정하였을 때의 응답시간

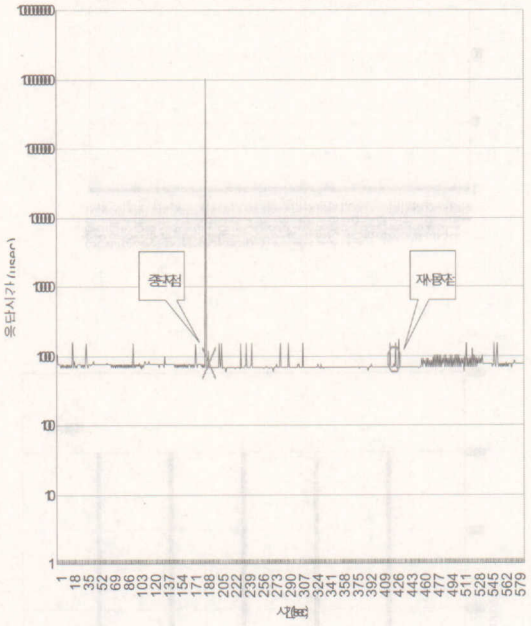


(그림 4.4) 하루에 전체 존 크기의 5%가 변화한다고 가정하였을 때의 응답시간

(그림 4.3)와 (그림 4.4)는 기존의 데이터로 서비스하고 있는 안정된 시스템에서 동적 갱신의 영향을 보여주고 있다. 동적 갱신의 영향으로 증가하는 응답 시간은 매스터 네임서버의 경우 대량의 추가 동적 갱신(그림 4.2)과 비교할 때 그 증가 시간이 유지되는 폭이 훨씬 줄어들어, 동적 갱신의 영향이 오래 지속되지 않음을 보였다. 슬레이브 네임서버의 경우, IXFR의 영향으로 매스터 네임서버보다 좀 더 오래 응답 시간의 지체가 계속되는 현상을 보였다. 실제 운용되는 도메인의 경우 동적 갱신의 규모가 하루를 기준으로 전체의 5% 이내가 될 것이라고 가정할 때, 고성능 시스템의 도입 및 적절한 시간 분배를 통해 갱신의 빈도를 높이면서도 만족할 만한 서비스 응답 시간을 보장하는 것이 가능하리라 판단한다.

4.2.2 이중화 환경에서 한 네임서버의 다운 및 재시동시 성능 측정

(그림 4.5)은 네임서버 중 하나의 기능을 중단시킨 후 다시 시동시켰을 때 응답시간 변화를 측정한 그래프이다. 이 그림에서 보여주듯이 중단 시점에만 응답시간에 지체가 있을 뿐 전체적인 응답 시간의 저하는 나타나지 않았다. 이유는 현재 한 개의 네임서버를 이용하여도 초당 1,000개의 질의는 충분히 처리할 수 있기 때문이다.



(그림 4.5) 매스터 네임서버 중 하나를 중단시키고 일정시간 후 다시 살렸을 경우

4.2.3. 동적 갱신의 두 부류인 ADD와 DELETE의 상대적인 성능 측정

다음으로 동적 갱신의 두 부류인 ADD와 DELETE[9]가 성능에 미치는 영향을 비교해 보았다. ADD와 DELETE를 포함하는 동적 갱신은 한번에 처리할 수 있는 양이 nsupdate의 구현상 2,500개가 한계이다. 그러나 이 수치는 네임서버의 처리 시간으로 인한 에러, 또는 컴퓨터의 사양, 존 파일의 크기 등에 영향을 받을 수 있다. 동적 갱신은 ADD를 할 때와 DELETE를 할 때 성능 차이가 심하게 난다. ADD의 경우 한번에 수행하는 개수가 약 2,500개를 넘지 않으면 정상적으로 수행이 되지

<표 4.2> 느린 서버에서 동적 갱신 한계 : 존 데이터의 개수가 500,000개 [한계:68개]

(단위: 초)

OP \ DU갯수	10	50	70	100	1,000	1,500	2,000
ADD	0.081090	0.118601	0.134865	0.162592	1.218184	1.879081	2.681495
DELETE	0.993801	4.687757	Error	Error	Error	Error	Error

<표 4.3> 빠른 서버에서 동적 갱신 한계 :존 데이터의 개수가 500,000개 [한계:410개]

(단위: 초)

OP \ DU갯수	100	200	300	400	500	1,000	1,500	2,000
ADD	0.157235	0.242293	0.358383	0.451116	0.566160	1.160295	1.770692	2.474160
DELETE	1.353779	2.577414	3.873643	5.165436	Error	Error	Error	Error

만, DELETE 작업은 동일한 환경에서 ADD보다 훨씬 적은 개수에서 에러가 발생하기 시작한다. 존 파일이 크거나 컴퓨터의 성능이 낮을 경우 한번에 DELETE 작업을 수행할 수 있는 양이 줄어들고, 이 양을 넘게 되면 에러가 발생한다. 또한 ADD에 비하여 DELETE 작업이 성능이 더 좋지 못하다.

<표 4.2>와 <표 4.3>은 각각 느린 컴퓨터 서버 및 빠른 컴퓨터 서버에서 동적 갱신을 수행했을 때 ADD와 DELETE 가능 개수를 표현한다. 현재 가지고 있는 존 파일의 데이터 수가 500,000개일 때 10개부터 2,000개까지 늘려 나가면서 ADD 및 DELETE 하는 시간을 측정하였다. 표에서 한계점이라고 표시한 것은 DELETE의 경우 동적 갱신의 수가 늘어나면서 에러가 발생하는 시점을 가리킨다.

존 데이터 및 ADD/DELETE 개수를 조절하면서 여러 번 실험을 해 본 결과, 위의 표와 유사한 결과를 얻을 수 있었다. 실험 결과로 볼 때 서버 컴퓨터가 빠르면 수행시간이 빨라짐은 물론, 한번에 DELETE 명령을 수행하는 개수도 증가한다. 그러므로 서버 컴퓨터와 존 파일의 크기에 따라 동적 갱신의 개수를 조절할 필요가 있다.

4.2.4 실험 내용 해석 및 실제 네임 서비스 시스템에의 적용

테스트베드의 컴퓨팅 환경이 실제 논문의 내용을 적용하여 운용할 환경과 다르기 때문에 위 실험 결과의 평면적인 적용은 의미가 없다. 다만, 실험을 통해 동적 갱신의 안정성을 보였고, 동적 갱신의 실험 데이터 크기와 그에 따른 실험 결과 부하로 판단할 때 적용 도메인의 규모가 kr 도메인의 수준이라면, 수용해야 할 데이터 양(초당 질의 수, 동적 갱신 데이터의 크기, 빈도 등)이 급격히 증가하지 않는 한 제안한 구조를 적절한 성능의 서버 시스템에 적용했을 때 충분히 감당할 수 있을 것이라고 판단된다.

위 실험에서의 결과를 보면 DELETE 작업의 수와 존 파일 데이터의 크기, 그리고 네임서버의 성능에 따라 전체적인 동적 갱신 작업이 영향을 받음을 알 수 있다. 따라서 시스템이 허락하는 한 동적 갱신을 위한 타이머의 간격을 줄이고, 정기적인 동적 갱신 작업은 작은 단위로 하는 것이 좋을 것이라 판단된다.

V. 고신뢰성을 위한 프로토콜

네임 서비스 시스템은 일반 컴퓨터 시스템과 달리 신속한 서비스 제공과 더불어 고도의 신뢰성(reliability)을 유지해야 한다. 이와 같은 요구 사항을 만족시키기 위하여 이중화된 매스터 네임서버를 기본 전제로 하여 하나의 서버에서 오류가 발생하더라도 다른 서버를 통하여 서비스를 계속할 수 있도록 한다. 매스터 네임서버의 이중화는 또한 서버로의 부하를 분담시키는 효과를 낼 수 있어 급증하

는 네임서버에의 성능 요구 사항을 만족시킬 수 있다는 두 가지의 의미를 동시에 지니고 있다. 본 장에서는 제안한 시스템 구조에서 고신뢰성을 달성하기 위한 각 구성 요소간의 프로토콜과 이를 이중화 서버의 동기 일치 방식에 대해 서술한다.

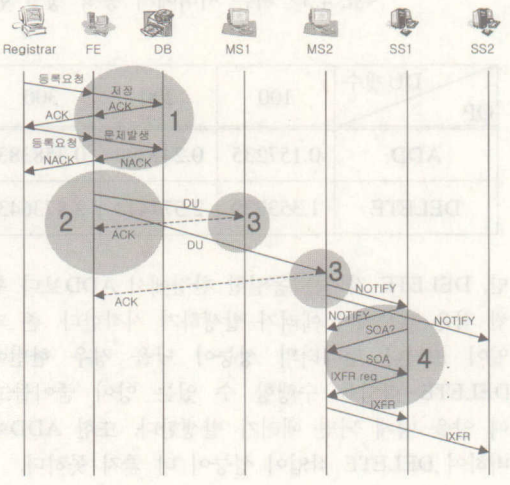
5.1 이중화된 매스터 네임서버 간 동기 (synchronization) 유지 방식

이중화된 매스터 네임서버는 양측이 동기를 맞추어 동일하게 구동시킴으로써 한 서버에서의 장애 발생 시 최대한 빠르게 장애 서버에서 제공하던 기능을 다른 네임서버에서 대신 수행할 수 있도록 한다. 동기의 단위는 네임서버 존 파일의 SOA SERIAL로서, 각 네임서버는 완전히 독립적으로 동작하지만 제공하는 서비스의 내용은 항상 같도록 조정한다. 이와 같은 방식은 FE에서 동적 갱신을 이중화된 네임서버 양측 모두에 수행한 후 자동적으로 동기화가 이루어진다는 이점이 있어, 동기화를 위한 부담을 최소로 줄일 수 있다.

매스터 네임서버는 이중화되어 있기 때문에 두 네임서버의 존 파일이 항상 동일한 내용을 유지하기 위해서는 FE로부터의 동적 갱신이 동시에, 또는 연속적으로 일어나 불일치의 기간을 최소화시키는 것이 필요하다. 동일한 존 파일에 기반한 서비스 제공은 사용자의 요구가 어떤 네임서버에 의해 수행될지 미리 알 수 없다는 것을 생각할 때 매우 중요한 요구 사항이다. 이를 보장하기 위해 FE는 주기적으로 이중화된 매스터 네임서버 각각에 메시지를 보내고, 각 매스터 네임서버는 자신의 SOA SERIAL을 포함하는 응답을 보낸다. 즉, 동기화의 단위는 동일한 존 파일을 의미하는 SOA SERIAL이고, FE는 수신한 두 SOA SERIAL이 동일한 경우 현재 이중화된 시스템의 상태를 normal로 판정한다. FE가 abnormal 상태를 판정하는 경우는 하나의 매스터 네임서버가 일시적인 오동작을 일으켜 SOA SERIAL을 제대로 갱신하지 못하였거나, 동적 갱신이 제대로 완료되지 않은 경우에 해당한다. 이때 FE는 SOA SERIAL이 작은 매스터 네임서버에 대해 존 DB를 이용하여 SOA SERIAL의 차이 부분에 해당하는 동적 갱신 내용을 수행한다. 이를 위해 존 DB에서는 각 매스터 네임서버 별로 동적 갱신이 완료된 것과 완료되지 않은 데이터를 구분하여 관리하며 DB에 SOA SERIAL을 함께 기록한다.

5.2 시스템 운용 흐름에 따른 문제 발생 위치 및 해결 방안

제안한 네임 서비스 시스템 전체 구조에서 각 모듈간의 인터페이스 및 프로토콜과 오류 발생 시 서비스에의 영향을 최소화하는 방안에 대해 서술한다. 앞으로 LB는 로드 밸런서(Load Balancer), FE는



(그림 5.1) 시스템 운용 흐름도

Front End, MS1 및 MS2는 각각 매스터 네임서버1과 2, SS는 임의의 슬레이브 네임서버를 의미하는 용어로 사용한다.

앞 장의 테스트베드 시스템에서 registrar를 통한 등록 요청부터, 슬레이브 네임서버에의 IXFR을 통한 데이터 전송까지 시스템 운용의 흐름이 (그림 5.1)에 나타나 있다.

(1) FE의 등록 과정 프로토콜

첫 번째로, (그림5.1)에서 ①로 표시된 부분으로 registrar로부터 등록된 데이터를 FE를 통하여 DB에 저장하는 과정에서 FE가 다운되거나 오동작 할 수가 있다. 먼저 데이터가 등록되는 과정을 보면, registrar가 FE에게 등록 요청을 하고 FE는 DB를 검색하여 올바른 데이터인지를 판별하고 올바를 경우 DB에 저장한 후 다시 registrar에게 ACK를 보내어 등록을 마치게 된다. 만약, 데이터가 중복되거나 올바르지 않을 경우 NACK를 보내어 오류가 발

생되었음을 registrar에게 알린다.

등록요청을 받은 데이터에 대해 FE는 registrar로부터 데이터를 받아 델타 DB 파일을 먼저 생성하고, 존 DB에 저장을 한 후 동적 갱신을 위한 델타 파일을 생성한다. 이러한 순서로 FE가 동작해야 하는 이유는 FE가 수행 도중에 갑자기 오동작 하거나 다운되는 경우에 대비하기 위해서이다. 실험 결과 FE가 registrar로부터 등록요청을 받고 파일 생성 및 쓰기 과정에서 오류가 발생한 후 복구가 되거나 또는 갑자기 FE가 멈추었다가 다시 동작할 경우, FE는 델타 DB파일에 있는 데이터가 존 DB에 저장이 되어 있는지, 그리고 델타 파일에 써져 있는지를 확인한다. 세 개의 파일에 동일한 데이터가 존재할 경우 파일 생성 및 쓰기 과정이 올바르게 종료되었다고 판단하고 registrar에게 응답을 보낸다. 만약 델타 DB파일에만 새로이 등록요청을 한 데이터가 존재하고 존 DB나 델타 파일에는 존재하지 않을 경우, FE는 존 DB와 델타 파일에 데이터 추가를 하고 registrar에게 응답을 보내는 것으로 복구과정을 마치게 된다.

(2) LB와 MS1 및 MS2 간의 프로토콜

LB에서는 적절하게 네임 서비스 요구를 MS1과 MS2에 분산시켜 특정 네임서버로 서비스 요구가 몰리지 않게 함으로써, 부하 분담의 효과를 제공한다. 부하를 분담하는 방식은 여러 가지가 있을 수 있으나, 현 상황에서는 단순히 특정 비율로 부하를 나누는 방식으로 충분히 원하는 목적을 달성할 수 있다고 판단된다. LB에서는 부하 분산을 위해 각 MS의 현재 상태를 감시하는 기능을 기본적으로 가지고 있어야 하며, 이를 통해 한 MS에 오류가 발생하여 정상적인 서비스 제공이 어려울 경우 타 MS가 일시적으로 서비스 요구를 전부 처리하도록 한다. 이와 같이 LB 자체의 판단에 의한 조치 이외에 FE에서의 명령에 따라 적절한 부하 조정을 수행한다.

(3) FE와 MS 간의 프로토콜

(그림 5.1)에서 ②로 표시된 부분으로 FE가 두 개의 매스터 네임서버로 동적 갱신을 하는 부분이다. FE는 이중화된 두 개의 MS로 nsupdate 함수를 이용하여 동적 갱신을 하게 된다. 이 때 FE는 두 개의 MS에 제대로 동적 갱신이 되었는지를 확인을 한 후에 자신의 델타 파일을 삭제하여, 동적 갱신 중 문제가 발생한 경우 이를 복구할 수 있도록 한다. 근본적으로 nsupdate가 동적 갱신이 종료되었음을 FE에게 알려주어야 하지만 현재의 nsudpate 함수에는 동적 갱신이 종료되었음을 FE에게 알려주는 기능이 없기 때문에, 여기서는 간접적으로 동적 갱신이 제대로 수행되었는지를 확인하는 방법을 사용한다. 확인 방법은 동적 갱신 후에 FE가 nslookup 함수를 이용하여 동적 갱신의 마지막 존 데이터에 대한 질의를 보내는 것이다. 동적 갱신은 ADD와 DELETE의 두 가지 모드가 존재하기 때문에 동적 갱신 된 마지막 데이터가 ADD를 위한 것이면 질의에 대한 응답이 되돌아오는 것을 확인하고, DELETE를 위한 것이라면 질의에 대한 응답이 오지 않는 것을 확인한다.

동적 갱신 수행 중 FE 또는 MS가 일시적으로 다운될 때 동작은 아래와 같이 구분한다.

가) FE의 MS로의 동적 갱신 수행 시 FE가 다운되는 경우

a. 동적 갱신 중간에 다운 - FE는 삭제되지 않은 델타 파일에 대해 MS에게 동적 갱신 을 다시 수행한 후 델타 파일을 삭제함으로써 동적 갱신을 완료할 수 있다.
b. 동적 갱신 후 델타 파일 삭제 전 다운 - 동일한 데이터를 지닌 삭제되지 않은 델타 파 일에 대한 동적 갱신을 재차 수행한다 할지라도 MS의 존 파일 내용은 변화가 없게 된다
c. 동적 갱신 주기의 중간 시점에서 다운 - MS의 서비스는 그 이전에 반영된 내용을 기반으로 중단 없이 수행된다.

나) FE의 MS로의 동적 갱신 수행 시 MS가 다운되는 경우

이 경우((그림 5.1)에서 ③ 부분) FE에서 nslookup을 이용한 검증이 실패하게 될 것이다. MS의 재시동은 최악의 경우 다운되기 이전의 존 파일 내용을 모두 유실했을 수 있기 때문에 현재까지 FE를 통해 갱신된 모든 DB 내용을 포함하도록 한다. 이를 위해 MS는 살아날 때 FE에게 최신의 내용을 반영한 전체 DB 파일을 요구하는 부분이 포함되어야 한다.

(4) MS와 SS간의 프로토콜

(그림 5.1)에서 ④ 부분은 MS에서 각 SS로 IXFR 전송시 발생할 수 있는 문제점을 나타내고 있다. MS는 FE로부터의 동적 갱신이 완료된 후 각 SS로 차례대로 IXFR을 수행한다. 우선 IXFR의 프로토콜을 살펴보면 MS에서 SS로 존 파일의 SOA SERIAL을 포함하는 NOTIFY를 보내고, 이를 받은 SS에서는 자신이 가지고 있는 존 파일의 SOA SERIAL을 포함하는 NOTIFY 응답을 MS로 전송한다. SS는 MS에서 전송해 온 SOA SERIAL을 확인하여 자신의 SOA SERIAL보다 최신의 것이라고 판단하면, 마치 refresh time 후에 수행하는 것처럼 SOA SERIAL을 MS에게 요청하고 MS는 SOA SERIAL을 응답한 후 SS에서 IXFR을 요청함으로써 IXFR을 수행하게 된다.

IXFR 전송 중 MS 또는 SS가 일시적으로 다운될 때 동작은 아래와 같이 구분한다. 어떠한 경우에도 서비스 중단은 해당 SS의 다운 타임 동안으로 한정되며 MS의 동작에는 영향을 미치지 않는다.

가) MS에서 SS로의 IXFR 수행 시 SS가 다운되는 경우

a. IXFR 도중 또는 이전에 SS가 다운 - MS는 IXFR 프로토콜에 의해 NOTIFY 응답, SOA 질의, IXFR 요구 중 어느 것 하나를 받지 못한 상태에서 SS로부터의 응답을 기다리게 된다. MS는 어느 시점 이후에 IXFR 수행을 중단하며, SS는 다시 동작을 시작할 때 MS로부터 존 파일을 AXFR을 통하여 받아오도록 한다.
b. IXFR 전송 중 또는 전송이 끝난 후 SS가 다운 - MS는 전송을 강제 중단시키고, SS는 다시 동작을 시작하면서 MS로 AXFR 존 파일 전송 요구를 하게 된다.

나) MSi에서 SS로의 IXFR 수행 시 MSi가 다운되는 경우

IXFR 대상 SS는 MSi로부터 SOA SERIAL 또는 IXFR 존 파일을 기다리던 상태에서 일정 기간 동안 응답이 없는 경우 IXFR 요구를 중지한다. 이 후 해당 SS는 이중화된 다른 MSj로 SOA 질의를 보냄으로써 IXFR 존 파일을 받기위한 시도를 한다. 이 MSj는 자신이 현재 동적 갱신을 받고 있는 상태인 경우 TRYLATER 메시지로서 응답한다. 이 메시지를 받은 SS는 일정 시점 이후 재시도 한다. 만약 MSj가 동적 갱신을 받고 있지 않다면 FE에 의해 이 MSj는 다운된 MSi와 SOA SERIAL의 측면에서 동일한 상태에 있기 때문에 SS에 IXFR을 수행하는 것이 허락된다. IXFR을 수행해야 할 MSi가 수행 이전에 다운되었다면, FE는 MSi가 정상이 아니라는 것이 감지하는 순간 MSj로 하여금 동적 갱신을 마치고 즉시 각 SS에 대한 IXFR을 수행하도록 지시한다.

Ⅵ. 결 론

인터넷의 DNS는 사실상 호스트 정보를 모아 놓은 데이터베이스라고 볼 수 있다. 사용자의 요구를 받아 적절한 정보를 반환해주는 간단한 역할을 하는 시스템이지만, 웹, 이메일, 파일 전송 등 모든 인터넷 응용이 가장 먼저 사용하는 인터넷에서 가장 중요한 서비스를 제공하고 있다. 따라서 네임 서비스 시스템의 효율적이고 안정적인 운용은 인터넷 서비스 제공에 있어서 매우 중요하다. 본 연구에서는 네임 서비스 제공을 위한 시스템에서 가장 중요한 역할을 하는 네임서버를 중심으로, 수용 능력 향상과 고신뢰성을 달성하기 위한 전체 시스템 구성 문제를 다루었다. 현재 우리나라에서 관리하는 대규모 도메인 관리를 전제로, 시스템 구조에 대해 여러 가지 대안을 제시하고 가장 단순한 형태로부터 쉽게 전이할 수 있는 효율적이며 신뢰성 있는 서버 시스템 구축을 제안하였다. 제안한 구조는 이중화된 네임서버를 가정하여 부하 분담 및 고신뢰성 달성이라는 두 가지 목표를 동시에 이루고자 하였다. 이를 위해 매스터 네임서버의 데이터 갱신을 위해 동적 갱신 방식을 사용하였고, 슬레이브 네임서버로의 데이터 전송을 위해 IXFR을 사용하여 서버의 부담을 최소화하고자 하였다. 테스트베드의 실험 결과를 실제 구축될 시스템에 적용할 경우, 향후 예상되는 서비스 관련 부하의 증가에 잘 대응할 수 있으며, 시스템의 다운이나 오동작의 경우에도 연속적인 서비스 수행이 가능하다고 판단된다.

대규모의 도메인을 관리하는 네임 서비스 시스템은 사회의 공공재로서 다루어져야 하므로 그 중요

성이 매우 크며, 따라서 시스템에 대한 전체 구조에 대한 연구 및 적용은 매우 신중하게 이루어져야 한다. 향후 더 높은 수준의 신뢰성 및 안정성과 지속적인 성능 요구가 반영될 경우, 제안한 구조가 더욱 개선될 필요가 생길 것이다. 하지만 부하의 폭증이 발생하지 않는 한, 제안한 구조를 따르는 시스템은 현시점에서 가장 단순한 구조에서 출발하여 최소의 비용으로 전이할 수 있는 가장 적합한 구조 중 하나라고 판단한다.

참 고 문 헌

[1] P. Mockapetris, "Domain Names - Concepts and Facilities," RFC 1034, November 1987

[2] A. Romao, "Tools for DNS debugging," RFC 1034, November 1994

[3] Cricket Liu, and Paul Albitz, DNS and BIND 4th Edition, O'Reilly, 2001

[4] http://www.powermct.co.kr/techED/tech/bind/dns-21.html

[5] http://www.domain.or.kr/

[6] http://www.isc.org/

[7] P. Mockapetris, "Domain Names - Implementation and Specifications," RFC 1035, November 1987

[8] R. Elz, R. and Bush, "Clarifications to the DNS Specification," RFC 2181, July 1997

[9] P. Vixie et al., "Dynamic Updates in the Domain Name System (DNS UPDATE)," RFC 2136, April 1997

[10] M. Ohta, "Incremental Zone Transfer in DNS," RFC 1995, August 1996

[11] P. Vixie, "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)," RFC 1996, August 1996

[12] http://www.nortelnetworks.com/products/01/alteon/webswitch/index_kr.html

[13] Eddie Harari, "BIND Version 8 Features," Linux Journal Volume 2000, Issue 69es, January 2000

[14] kldp.org/KoreanDoc/html/PoweredByDNS-KLDP/nslookup.html

[15] W. R. Stevens, UNIX Network Programming, Prentice-Hall, 1998

심 영 철(Young-Chul Shim)

1979년 2월 : 서울대학교 전자공학과 학사

1981년 2월 : 한국과학기술원 전기 및 전자공학과 석사

1991년 12월 : University of California, Berkeley, 전산학박사

1993년 2월 - : 홍익대학교 정보컴퓨터공학부 부교수

<관심분야> 인터넷 프로토콜, 보안, 클러스터 컴퓨팅

박 준 철(Jun-Cheol Park)

1986년 서울대학교 계산통계학과(학사)

1988년 KAIST 전산학과(석사)

1998년 미국 Maryland 대학교 전산학과(박사)

현재 홍익대학교 정보컴퓨터공학부 조교수

<관심분야> Mobile Wireless Internet, Active Networks

강 호 석(Ho-Seok Kang)

2000년 홍익대학교 컴퓨터정보통신(학사)

2002년 홍익대학교 전자계산학(석사)

현재 홍익대학교 컴퓨터공학과 박사과정

<관심분야> 인터넷 보안, 모바일 인터넷, 클러스터 컴퓨팅

이 준 원(Jun-Won Lee)
2002년 홍익대학교 컴퓨터 공학(학사),
2004년 홍익대학교 컴퓨터 공학(석사)
현재 에이엠텍 코리아 시스템
개발팀 사원

<관심분야> 인터넷 보안, 모바일 인터넷