교육학석사학위논문

파일 타입에 따른 웹 캐쉬 분할기법

지도교수 윤 성 대

이 논문을 교육학석사학위논문으로 제출함

부경대학교 교육대학원

전 산 교 육 전 공

오 윤 주

오윤주의 교육학석사 학위논문을 인준함

2003년 6월 21일

주　　심　공학박사　여 정 모 (인)

위　　원　이학박사　박 홍 복 (인)

위　　원　이학박사　윤 성 대 (인)

<차 례>

<그림 차례>

<표 차례>

A Web Cache Partition Strategy based on File Type

Yun-Ju Oh

Graduate School of Education
Pukyong National University

Abstract

As the World Wide Web becomes more and more popular in recently, the number of WWW users is increasing rapidly. Therefore the caching of the internet object became problem that is very important. Web cache must handle objects that have various kind and size, because it is different from the traditional cache.

To gain a good hit rate in web Cache, several access methods are studied. Common issue of all the caching approaches is to manages finite space of web cache efficiently. Therefore a basic issue of a caching scheme is how to manage its finite storage space in order to obtain good hit rates.

In this paper, we propose a partitioned web cache based on file type to manage web cache space and use efficient replace algorithm in each class. To correspond in user's object request change, the partition rate is update in fixed

cycle. In simulation, we used an open trace in internet to compare the existent policy with the proposed policy. As a result, the proposed policy showed better performance than the existent policy.

Ⅰ. 서 론

최근 인터넷 사용자와 서비스 제공자의 기하급수적인 증가에 따라 동일 객체에 대한 중복 요청 횟수가 증가하고 있다. 이로 인해 네트워크 대역폭이 불필요하게 낭비되며, 일부 인기 있는 서버로 부하가 집중되어 클라이언트의 지연 시간이 길어지거나 접속 불능이 되는 상태까지 발생하고 있는 실정이다. 이들 문제는 대중적인 파일의 복사본을 서버에서 사용자에게 가까운 지점으로 보냄으로써 완화할 수 있는데 가장 일반적인 방법[1]이 캐싱이다.

캐싱은 서버의 부하를 줄이고, 웹 문서의 요청들로 인해 발생되는 네트워크 트래픽을 줄이는데 도움이 된다. 그리고 캐싱된 문서는 웹 사용자들에게 빠른 응답시간을 제공하여 결과적으로 전체적인 웹의 성능을 향상시킬 수 있다.

인터넷에는 다양한 파일 타입이 존재하며 파일 타입마다 평균 크기, 접근 횟수 등이 다르다. 예를 들어 인터넷 객체의 크기는 수 Byte에서 수십 Mega Byte까지 다양하며 각 객체의 인기도 또한 매우 가변적[2]이다. 따라서 인기가 없는 매우 큰 객체의 삽입으로 인하여 크기는 작으나 매우 인기 있는 수많은 객체들이 제거되어야 하는 상황이 발생할 수 있다. 이 경우 인터넷 캐싱의 효율은 매우 떨어지게 된다. 그러므로 웹 캐쉬는 전통적인 캐쉬와는 달리 다양한 종류와 크기의 인터넷 객

체를 지원해야 한다.

웹 캐쉬는 클라이언트 캐쉬, 서버 캐쉬, 프락시 캐쉬, 계층적 캐쉬, 협력적 캐쉬[6] 등 여러 접근 방법이 있다. 모든 접근방법들은 보다 나은 적중률을 얻기 위한 공통의 문제로 웹 캐쉬의 한정된 저장 공간을 어떻게 효율적으로 이런 관리할 것인가에 초점을 두고 있다.

이에 대한 접근방식의 하나로 새로운 문서를 위해 저장된 문서를 교체하는 교체 알고리즘에 대한 연구가 있으며, FIFO(First In First Out), LRU(Least Recently Used), LFU(Least Frequently Used), SIZE 등의 많은 교체 알고리즘의 연구[1,7]가 진행되어져 왔다. 또한 기존 알고리즘 외에도 최근 LRFU(Least Recently/Frequently Used) 교체 알고리즘[8]이 연구되었는데, 이 알고리즘은 효율성이 대단히 우수한 것으로 알려져 있다.

또 다른 접근 방법으로 웹 환경에서 인식되는 높은 변동성을 고려하면서 캐쉬 공간 관리의 효율성을 높이기 위한 분할 캐쉬(Partitioned Cache) 방식[1,5]이 있다. 기존의 분할 캐쉬에 대한 실험결과는 캐쉬 저장 공간을 관리하기 위한 분할 캐쉬 방식이 웹 문서의 다양성을 고려하기 때문에 캐쉬 공간을 분할하지 않는 교체 알고리즘보다 낫다[5]는 것을 보여주고 있다.

분할 캐쉬는 캐쉬 저장 공간을 여러 개의 분할된 영역으로 나눔으로써, 이질적인 웹 상의 객체를 동종의 데이터 집합으로 나누어서 각각의 분할 영역에서 다루도록 할 수 있게 한다. 이

는 각각 같은 특성을 갖고 있는 데이터 집합의 참조 지역성을 분할 영역 안에서 보존하기 위함이다. 그러나 캐쉬 공간을 특정 비율로 고정시켜 분할하는 방법은 분할된 각 영역에 같은 교체 알고리즘을 적용함으로써 분할된 캐쉬의 장점을 제대로 활용하지 못한다는 단점이 있다.

그러므로 본 논문에서는 캐쉬를 분할했지만 각 영역에 같은 교체 알고리즘을 적용함으로서 발생하는 단점을 극복하기 위하여, 캐쉬 공간을 파일 타입별 클래스로 분할하고 서로 다른 교체 알고리즘을 적용하며, 웹 사용자들의 객체 요구 변화에 대응하기 위하여 주기적으로 캐쉬 클래스의 분할비를 갱신하는 효율적인 분할 캐싱 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 일반적인 캐쉬의 동작 원리와 기존의 캐싱 기법에 대해 살펴보고, 3장에서는 기존 알고리즘의 단점을 해결하기 위한 분할된 캐싱 기법을 제안한다. 4장에서는 실험 결과 및 성능평가를 보이고, 마지막으로 5장에서는 결론 및 향후연구 과제에 대해 기술한다.

Ⅱ. 관련연구

이 장에서는 관련연구로 웹 캐쉬의 구성요소 및 교체 알고리즘에 대해 살펴보고, 분할 캐쉬 기법과 캐쉬 성능 평가 방법에 대해 설명하고 있다.

2.1 일반적인 구성 요소 및 동작원리

인터넷 캐싱 시스템의 객체들은 클라이언트 브라우저, 프락시 서버, 그리고 원본 서버에 캐쉬 될 수 있다. 클라이언트는 요청한 문서가 브라우저의 캐쉬에 존재하지 않으면 프락시 서버에 요청한다. 클라이언트로부터 요청을 받은 프락시 서버는 요청된 객체가 캐쉬 되어있는지를 검사하고 서버에 존재하면, 클라이언트에 서비스를 제공한다. 만약 존재하지 않으면, 프락시 서버는 원본 서버에 요구하여 전송 받은 객체를 자신의 캐쉬 공간에 저장한 후 클라이언트에 제공하는 일반적인 작동구조[10]를 가진다.

현재 몇 개의 시제품이나 프로토타입이 있으며, 하비스트 캐쉬 서버는 콜로라도 대학에서 만들어진 시스템이다. 부모와 형제를 갖는 계층적 구조를 가지며 이들 간의 상호협력은 UDP(User Datagram Protocol)를 기반으로 한 ICP(Internet

Cache Protocol)[11]로 이루어진다. 현재 하비스트는 상업적인 목적으로 안정성과 성능을 대폭 향상시킨 하비스트 제품군과 좀더 지능적인 캐쉬를 구현하기 위해 새로운 기능을 계속 추가하고, 연구를 위해 원시코드가 공개된 스퀴드 인터넷 캐싱 시스템으로 구분된다.

2.2 Web Cache 교체 알고리즘

캐싱 기법의 성능은 한정된 캐쉬 공간 하에서 캐쉬에 보관하고 있을 객체와 삭제할 객체를 동적으로 결정하는 온라인 기법인 캐쉬 교체 알고리즘에 의해 크게 좌우된다. 캐쉬 교체 알고리즘은 미래의 참조를 알지 못하는 상태에서 미래에 참조될 가능성이 높은 객체를 선별하여 한정된 캐쉬 공간에 보관하는 것이므로 그 효율성은 궁극적으로 미래의 참조 가능성을 얼마나 잘 예측하는가에 좌우된다. 현재 많은 캐쉬 교체 알고리즘이 제안되었고 각각의 성능평가가 이루어졌다. 캐쉬 교체 알고리즘에는 다음과 같은 것들이 있다.

(1) FIFO(First In First Out)

캐쉬에 들어온 시간순으로 객체를 교체하는 알고리즘이다.

객체의 중요도나 다른 가중치를 설정하지 않고 오래된 순서대로 교체를 한다. 자주 사용되고 많이 사용되는 객체도 시간이 지나면 교체될 수 있는 단점이 있다.

(2) LRU(Least Recently Used)

LRU는 캐쉬내의 객체들 중 최근 참조된 시각이 가장 오래된 객체를 교체하는 알고리즘이다. 이 알고리즘은 시간적 지역성(temporary locality)에 기반한 알고리즘으로 최근에 사용된 객체가 가까운 시간 내에 다시 사용될 가능성이 높다는 점을 이용한다.

LRU의 주요 단점중의 하나는 한번만 접근되는 객체는 캐쉬에 저장될 필요가 없음에도 불구하고 이들이 오랫동안 캐쉬에 남아 있다는 점이다. 즉, 오직 한번 참조된 객체에 의해서 재사용 될 가능성이 높은 객체가 교체될 수도 있다는 것이다. 이는 LRU가 시간적 지역성만을 반영하고 접근회수를 반영하지 않기 때문에 발생하는 문제이다.

일반적으로 LRU 교체 알고리즘은 하드웨어나 다른 캐쉬 시스템에서는 우수한 성능을 가지고 있으나 인터넷 캐쉬에서는 다른 교체 알고리즘에 비해 크게 우수한 성능을 갖지 못한다 [1,3].

(3) LFU(Least Frequently Used)

LFU는 참조 횟수가 가장 작은 객체를 삭제하는 교체 알고리즘으로서, 많이 참조된 객체는 계속해서 참조 될 가능성이 높다는 점에 기반한 알고리즘이다. 그러나 LFU를 사용하면 처음에 많이 사용되어 참조 횟수는 많지만 나중에 더 이상 사용되지 않는 객체가 캐쉬에서 제거되지 않는 경우가 발생한다. 이는 이른바 "캐쉬 오염(cache pollution)" 현상[3]의 원인이 되고, 유효 캐쉬 크기의 감소를 일으킨다.

인터넷 캐쉬에서는 사용시간에 기반한 LRU 교체 알고리즘보다 사용 빈도수에 기반한 LFU 교체 알고리즘이 보다 나은 성능[9]을 보인다.

(4) SIZE

SIZE는 크기가 가장 큰 객체를 삭제하는 교체 알고리즘이다. 인터넷 캐쉬는 하드웨어 캐쉬나 파일 시스템 캐쉬와는 다르게 캐쉬의 단위가 인터넷 객체이기 때문에 캐쉬 할 객체의 크기가 가변적이다. 따라서 크기가 큰 하나의 객체가 크기가 작은 여러 개의 객체들을 교체하는 경우가 종종 발생한다. 이 알고리즘은 캐쉬 된 객체 중에서 크기가 가장 큰 객체를 먼저 교체함으로써 이러한 문제점을 개선한다.

SIZE 교체 알고리즘은 인터넷 캐쉬에서 우수한 성능을 갖는

것으로 평가[1]되고 있다.

(5) LRFU(Least Recently/Frequently Used)

LRFU 교체 알고리즘은 각각의 참조 시점을 그 최근성에 근거하여 고려하는 방법으로, 과거 시점에서의 각각의 참조가 현재 객체의 가치를 높이는 데에 기여하는 바를 독립적으로 계산하여 더하는데 최근의 참조일수록 가치 기여도를 더 높게 계산한다. 이 알고리즘은 LRU와 LFU를 이용하면서도 웹에 적용하기 위해 몇 가지 단점을 보완한 교체 알고리즘[8]이다.

예를 들어 그림 1의 (a)에서 과거 참조 기록을 바탕으로 객체 A와 B를 평가한다고 하자. LRU 정책은 객체 B가 좀 더 최근(t_1)에 참조되었기 때문에 B에게 더 높은 가치를 부여한다. 하지만 이 방법은 가장 최근 참조기록만을 활용하기 때문에 자주 참조되는 문서와 그렇지 않은 문서를 구분할 수 없다는 단점이 있다. 그림 1의 (b)에서 A의 가치가 더 높게 평가된다는 사실은 LRU 알고리즘의 문제점을 단적으로 보여주고 있다.

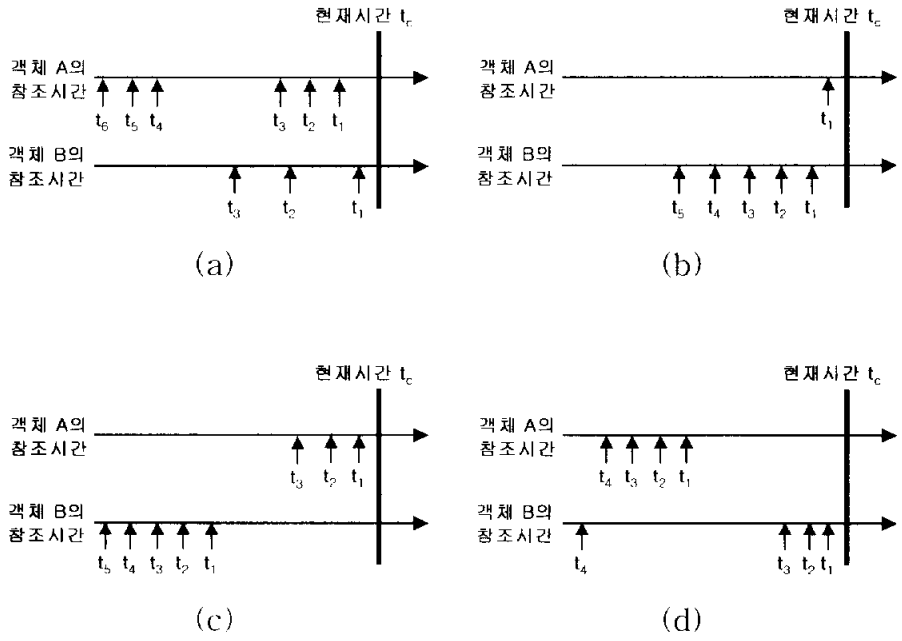

그림 1. 과거 참조 기록에 의한 객체의 가치 평가

LFU 알고리즘은 참조 회수가 가장 적은 객체를 캐쉬에서 삭제한다. 그림 1의 (a)에서 LFU 알고리즘은 객체 A의 참조 회수가 B보다 더 많기 때문에 A에 더 높은 가치를 부여한다. 이 방법의 문제점은 최근에 전혀 참조되지 않았더라도 오래 전에 많이 참조된 객체에 높은 가치를 부여하여 새로 참조되기 시작한 객체를 캐쉬에서 삭제할 우려가 있다는 점이다. 그림 1의 (c)에서 B의 가치가 더 높게 평가된다는 사실은 이러한 방법의 문제점을 보여주고 있다.

LRU-k는 최근 k번 참조된 시간에 의거하여 가치를 평가한다. k=3이라 할 때 그림 1의 (a)에서 웹 문서 A의 t_3가 현재

시간에 더 가깝기 때문에 이 방법에 의하면 A의 가치를 더 높게 평가하게 된다. 하지만, 이 방법은 최근 k-1번 참조된 시간은 무시되고 k번째 참조 시간만이 고려되기 때문에 최근 참조 성향을 정규화된 방법으로 반영하지 못한다는 단점이 있다. 그림 1의 (d)에서 k=4인 경우 A의 t_4가 현재 시간에 더 가깝기 때문에 A의 가치가 더 높게 평가된다는 사실은 이러한 방법의 문제점을 보여주는 극단적인 예이다.

LRFU 알고리즘은 이러한 방법들의 문제점을 극복하여 과거 참조 기록에 의한 가치 평가 부분을 참조 빈도와 최근 참조 성향을 정규화된 방법으로 반영하는 기법이다. 이 기법은 과거 시점에서의 각각의 참조가 현재 객체의 가치를 높이는데 기여하는 바를 따로 계산하여 더하는 방법으로 참조 시점이 현재로부터 멀어질수록 가치 기여도도 낮아지게 된다.

기존의 LRU 방법이나 LFU 방법도 최근성과 최빈성을 이용해서 교체가 이루어질 객체를 찾아내어 객체 교체시 로드(load)를 감소시키는 방법인데, 이 두 기법과의 차이점은 기존의 LRU 기법과는 다르게 객체의 부재비용에 관계없이 모든 페이지 정보를 유지해야하는 비용(overhead)을 피하려는 것이고, LFU 기법과는 다르게 캐쉬 오염을 피하려는 것이다. 이렇게 되면 기존의 LRU나 LFU 방법보다는 제어변수를 이용하는데 따른 추가비용이 많아지겠지만 객체의 교체 시간을 최소한으로 줄일 수 있어 전체 수행시간이 짧아진다.

LRFU 알고리즘은 다소 복잡해 보이는 가치 평가 방법에도 불구하고, 시간 복잡도 측면에서는 한번의 캐쉬 연산에 드는

시간이 $O(\log_2 n)$이며, 공간 복잡도 측면에서는 캐쉬 내의 객체 당 부가적으로 저장되는 정보가 O(1)로서 그 효율성이 대단히 우수한 것[8]으로 알려져 있다.

2.3 분할 캐쉬

분할 캐쉬란 모든 객체를 저장하는 단일 캐쉬를 가지는 것 대신, 캐쉬 저장 공간을 여러 개의 분할 영역으로 나누어서 객체들의 크기나 타입에 따라 각각의 객체들을 분할하여 저장하는 방법[5]이다.

분할 캐쉬의 사용은 메모리와 처리기 사이의 대역폭을 증가시키기 위해 이미 컴퓨터 구조분야에서 사용되고 있다. 예를 들어 두 영역으로 나누어진 캐쉬의 한 영역에는 명령어들을 위한 부분으로 사용되고 나머지 한 영역은 데이터를 위한 부분으로 사용하는데, 이런 형태는 현재의 대부분 처리기에서도 사용되고 있다. 이와 같은 컴퓨터 구조에서의 분할 캐쉬는 명령어 캐쉬가 데이터 캐쉬에 비해 더 높은 적중률을 갖는다고 조사[12]되고 있다.

웹에서의 분할 캐쉬는 인식되는 객체 크기의 다양성을 고려하기 위해 캐쉬 저장 공간을 관리하기 위한 방법으로 제안[1]되었다. 웹에서 분할 캐쉬를 설계하기 위해서는 캐쉬를 몇 개

로 분할할 것인가, 각 분할 영역의 크기, 각 분할 영역에 포함될 파일의 종류, 각 영역에 사용할 교체 알고리즘 등 몇 가지 고려해야 될 요소가 있다.

각각의 분할 영역은 객체 크기나 타입에 의해서 분류되어지므로, 분할된 캐쉬 공간은 같은 종류의 객체들을 관리할 수 있게 되며 그에 맞는 교체 알고리즘의 적용이 가능해진다. 캐쉬를 분할하는 근본적인 이유는 각각의 분할 영역에서 참조 지역성을 보존함으로써 전체 캐쉬 공간의 적중률을 높이고자 하는 데 있다. 특정 분할 영역에 저장된 객체는 오직 그 객체와 같은 종류의 객체에 의해서만 교체되므로, 크기가 작은 객체의 참조 지역성이 크기가 큰 객체 때문에 방해되어지지 않게 된다. 이는 일반적으로 크기가 작은 객체가 상대적으로 높은 요청빈도를 가지고 있다는 사실에 근거하고 있다.

웹을 위한 분할 캐쉬 연구는 Stephen Williams가 처음 제안하였다. 그러나 이 연구에서는 단순히 audio와 non-audio의 2가지 타입의 객체를 대상으로 연구되었다. 분할기법에 관한 다른 연구로는 파일 타입 기반 캐싱 기법[15]과 크기에 따른 캐쉬 분할 기법[16]이 있다. 파일 타입을 기반으로 캐싱한 연구의 경우 3가지 파일 타입으로 캐쉬를 분할하고, 각 분할캐쉬에 동일한 교체정책을 사용하여 통합된 캐쉬보다 적중률과 바이트 적중률 모두에서 더 좋은 성능을 보여준다는 것을 지적[15]하고 있다. 하지만 이들 연구는 웹 캐쉬를 분할했지만 각 영역에 동일한 교체정책을 사용하여 분할캐쉬의 최적의 성능을 보여주지는 못하였다.

2.4 캐쉬 성능 평가 방법

캐쉬 성능은 적중률(hit rate : HR)과 바이트 적중률(byte hit rate : BHR)로 평가[14]한다. 캐쉬 적중률은 사용자의 전체 요청 수에서, 캐쉬에 파일이 존재하여 적중된 수의 비율을 의미한다. 인터넷 캐쉬에서 캐쉬 적중률(HR)은 서버에 도착하는 사용자 요청의 감소비율을 나타낸다.

인터넷 캐쉬는 고정된 크기의 페이지 단위로 캐쉬를 하는 것이 아니라 가변 길이의 파일 단위로 캐쉬를 하기 때문에 단순히 캐쉬 적중률만으로 평가하기는 어렵다. 이를 위해 사용자가 요청한 전체 크기에서 적중한 객체 크기 양의 비율로 나타내는 바이트 적중률을 이용한다.

인터넷 캐쉬에서 바이트 적중률(BHR)은 캐쉬를 사용함으로써 얻어지는 네트워크 트래픽 감소와 응답 시간 향상을 나타내는데 사용된다. 하지만 인터넷의 특수성 때문에 응답시간의 향상은 예측할 수 있으나, 정확하게 정량화 하는 것은 불가능하다. 왜냐하면, 인터넷에서의 응답속도는 객체의 크기뿐만 아니라 지리적 혹은 네트워크의 트래픽 상황에 따라 크게 영향을 받기 때문이다.

캐쉬 적중률에 대한 정의는 식 (1)과 같다.

$$HR = \frac{\sum_{i=1}^{n} hit_i}{\sum_{i=1}^{n} req_i} \qquad (1)$$

바이트 적중률에 대한 정의는 식 (2)과 같다.

$$\mathrm{BHR} = \frac{\sum_{i=1}^{n} b_i hit_i}{\sum_{i=1}^{n} b_i req_i} \qquad (2)$$

여기서, b_i는 i번째 객체의 크기이고, hit_i는 i번째 객체를 캐쉬로부터 응답한 횟수, 그리고 req_i는 i번째 객체가 요청된 횟수이다.

III. 파일 타입에 따른 분할캐쉬 기법

이 장에서는 본 논문에서 제안하는 파일 타입에 따른 분할 캐쉬 기법을 기술하고, 파일 타입별 캐쉬의 크기 및 제안한 웹 캐쉬의 동작 구조를 설명하고 있다.

3.1 파일 타입에 따른 분할캐쉬 기법

인터넷에서는 다양한 타입의 파일들이 존재하고 타입마다 가지고 있는 특징이 다르다. 그러나 지금까지의 인터넷 캐쉬의 캐싱 전략은 파일을 타입에 따라 구분하지 않고 모든 파일을 동일하게 취급함으로써 캐싱의 비효율성 문제를 발생시켰다.

예를 들어 타입별로 구분되는 특징 중에 하나인, 파일 크기가 다른 경우를 고려해보자. 텍스트 파일의 평균 크기는 1, 비디오 파일의 평균 크기는 5, 캐쉬의 크기는 10이라고 하고 교체 알고리즘은 FIFO를 사용해보자.

그림 2에서 보는 것과 같이, 파일 타입에 관계없이 캐쉬 내에서 동일하게 취급하는 경우, 크기가 큰 비디오 파일이 크기가 작은 여러 개의 텍스트 파일을 제거함으로써 캐쉬 적중률을 감소시키는 일이 발생할 수 있다.

또 다른 경우를 보면 그림 2의 상태 이후에 텍스트 파일이

계속 요청되면 텍스트 파일에 의해 비디오 파일이 교체된다. 비디오 파일이 캐쉬에서 적중되면, 비디오 파일은 다른 타입의 파일에 비해서 크기가 크기 때문에 바이트 적중률은 상대적으로 많이 증가한다. 그러나 LFU, SIZE 교체 알고리즘을 사용하면 접근 횟수가 적고, 크기가 큰 비디오 파일은 텍스트에 의해 자주 교체된다.

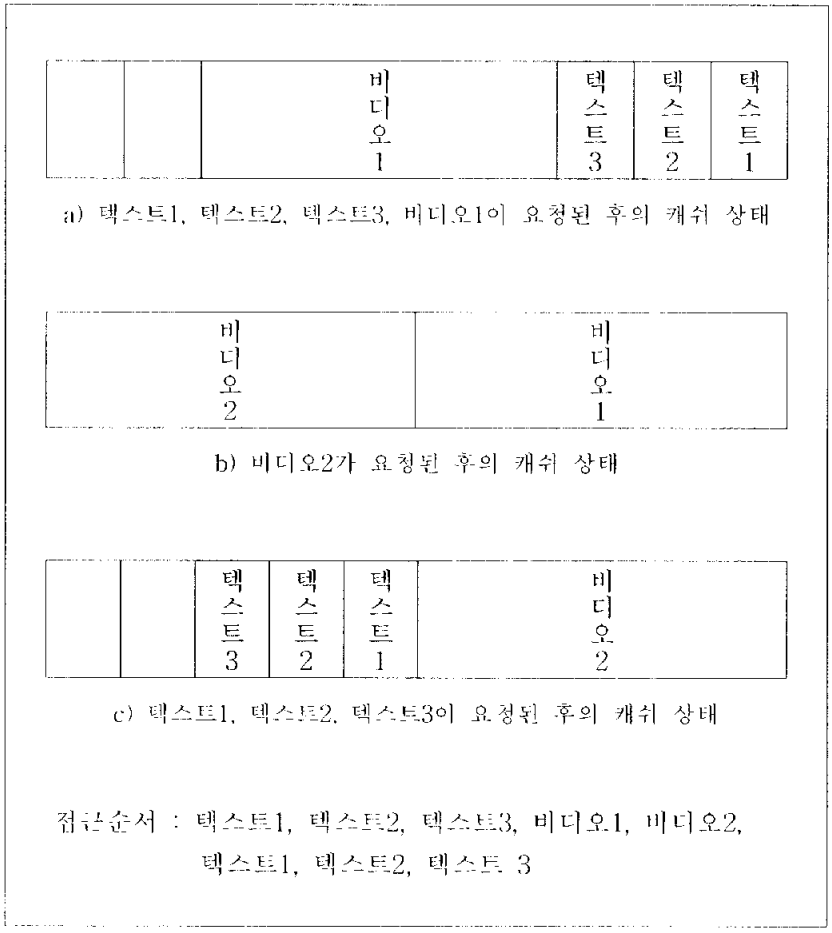

그림 2. 통합된 캐쉬에서의 교체 과정의 예

본 논문에서는 이런 문제를 해결하기 위해서 파일 타입별 클래스로 분할된 캐쉬구조를 제안한다. 이는 웹에서 요청되는 객체가 같은 파일 타입이면 크기 또한 비교적 비슷한 분포를 갖는다는 Martin F.Arlitt의 연구[4]를 바탕으로 한다. 즉 모든 객체를 저장할 단일 캐쉬 공간을 갖는 대신에, 비교적 비슷한 크기를 갖는 파일 타입에 기초한 객체의 클래스들을 저장할 몇 개의 구역을 갖는 분할된 캐쉬 구조를 제안한다.

각각의 클래스는 비슷한 사이즈를 갖는 파일타입에 의해 구분된다. 비교적 요청 횟수가 높지만 평균 크기가 작은 파일 타입이 요청 횟수는 낮지만 사이즈가 매우 큰 파일 타입에 의해 희생당하지 않을 일정한 영역을 할당함으로써 비교적 높은 캐쉬 적중률을 보장받을 수 있다.

그림 3은 본 논문에서 제안하는 파일 타입별 캐쉬에서의 교체 과정 예이다. 캐쉬를 파일 타입별로 구분하여 그림 2와 동일하게 접근 순서를 적용하면 3개의 파일이 캐쉬 내에 적중할 수 있다.

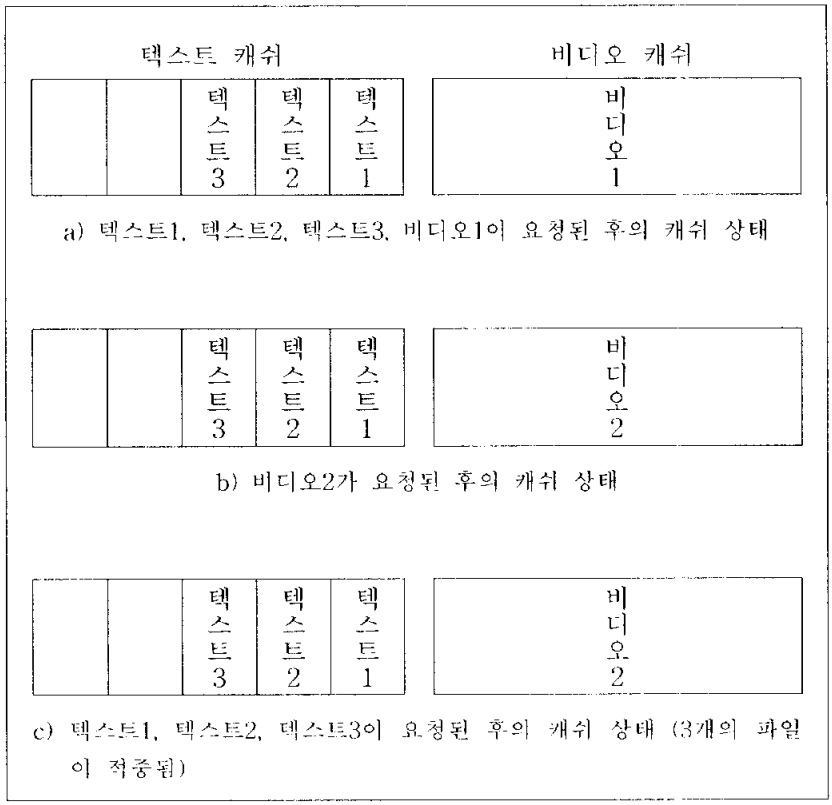

그림 3. 타입별 캐쉬에서의 교체 과정의 예

3.2 파일 타입별 캐쉬 크기

인터넷 객체의 종류에 따른 요구율과 객체별 전송량 비율을 분석해놓은 여러 연구들[1,4]을 살펴보면, Image 파일 타입의 객체들은 일반적으로 50%에서 70%사이의 요구율을 보이고 있다. 비록 Image 파일 타입 객체 각각의 크기는 작을지라도 요

청되는 횟수가 많기 때문에 총 전송된 크기에서 차지하는 비율은 높다는 것을 알 수 있다.

Html 파일 타입의 객체들도 약 30%에서 40%정도의 요구율을 보이고 있으며 총 전송된 크기에서도 Image 파일 타입 다음으로 많다는 것을 알 수 있다. 또한 멀티미디어 타입의 객체들의 요구율은 아주 미약하지만 요구율에 비하면 총 전송된 크기에서 차지하는 비율은 상대적으로 높다. 즉, 크기가 작은 객체들을 많이 요청함으로써 네트워크 트래픽이 증가할 수 있지만, 비디오 파일 타입이나 오디오 파일 타입과 같은 크기가 큰 객체는 몇 번만 요청해도 통신량이 많이 증가함을 의미한다.

이 연구들의 분석 결과가 다른 서버의 작업 부하와 일치한다고 단정할 수는 없다. 특정 인터넷 서버를 이용하는 사용자의 경향이나 서버의 역할에 따라 다르기 때문이다. 하지만 특수한 목적을 가진 서버가 아닌 이상은 위의 연구 결과들과 비슷한 경향을 보인다.

본 논문에서는 캐쉬 공간을 Graphic, Text, Etc의 3개 클래스 영역으로 분할한다. Graphic 클래스에는 gif, jpg, jpeg 등의 확장자를 가진 파일이 포함되고, Text 클래스의 경우는 txt, html, htm와 같은 확장자를 가진 파일이 포함된다. 그리고 Etc 클래스에는 그 외의 video, audio 등의 여러 파일과 알려지지 않은 확장자가 포함된다.

분할 캐쉬에서 하나의 통합된 캐쉬를 여러 개의 캐쉬로 구분할 때 어떤 비율로 구분할 것인가는 중요한 문제이다. 캐쉬를 각각의 클래스로 분할할 때는 각 클래스가 일정한 영역을

보장받을 수 있도록 한다. 즉, 사용자가 많이 사용하는 타입에는 캐쉬의 많은 부분을 할당하고, 그렇지 않은 타입에 대해서는 적게 할당해야 한다.

각 클래스별 분할비(%)는 식 (3)과 같다.

$$\text{클래스별 분할비}(\%) = \frac{\text{클래스별 전송량}}{\text{전체 전송량}} \qquad (3)$$

또한 본 논문에서는 사용자의 요구를 동적으로 반영하기 위해서 클래스별 분할비를 주기적으로 변경시킨다. 웹 사용자는 매번 정해진 비율로 객체를 요청하는 것이 아니고, 요청하는 객체 타입들의 비율이 계속 바뀌기 때문에 주기는 일반적인 활동 주기인 1주일로 정한다.

클래스별로 분할된 캐쉬는 각 분할영역에 맞는 교체 알고리즘을 적용한다. 즉, LRU, LFU, SIZE, LRFU를 각각의 분할캐쉬에 한번씩 적용시켜 좀더 나은 성능을 보이는 교체 알고리즘을 선택한다. 이때 교체 알고리즘의 성능을 나타내는 적중률(hit rate)과 바이트 적중률(byte hit rate) 중 어느 쪽을 기준으로 하느냐에 따라 결과가 달라질 수 있는데, 두 가지를 모두 적절히 고려하여 교체 알고리즘을 선택하기로 한다.

3.3 캐쉬의 동작 구조

본 논문에서 제안한 캐쉬의 동작구조는 다음과 같다.

캐쉬 관리자는 캐쉬에 저장된 인터넷 객체의 리스트를 관리한다. 사용자의 특정 객체에 대한 요구가 들어오면, 캐쉬 관리자는 요구된 객체를 리스트에서 검색한다. 만일 적중하면 요구된 객체를 인터넷 캐쉬로부터 클라이언트로 전송한다. 그렇지 않으면, 관리자는 원본이 있는 인터넷 서버에 요청을 한다.

이 동작으로 객체가 전송될 것이고, 전송 받은 객체는 그 파일 타입에 따라서 해당하는 캐쉬 클래스에 저장된다. 만약 전송 받은 인터넷 객체를 위한 공간이 없을 경우에는, 바로 그 영역에서 여유공간이 생길 때까지 해당하는 교체 알고리즘을 이용하여 삭제할 객체를 선택하게 된다.

제안한 캐쉬의 동작 구조 알고리즘은 그림 4에 나타내고 있다.

```
Set each partition size by % of byte transferred
Initialize each partition cache
Initialize theclass ← 0
Process each object in turn
    the current request is for object p
    theclass ← objectNumber(object type(p))
    if p is already in theclass
            hit_cache(p, theclass)
    if p is not in theclass
            if theclass is 1 or 2
                    lrfu_replace(p, theclass)
            if theclass is 3
                    size_replace(p, theclass)
End
```

그림 4. 제안한 캐쉬의 동작 구조 알고리즘

Ⅳ. 파일 타입에 따른 분할캐쉬 기법의 분석 및 성능 평가

이 장에서는 시뮬레이션에 사용된 trace의 특징 분석 및 제안한 파일 타입에 따른 분할캐쉬 캐쉬기법의 시뮬레이션 결과를 나타내고 있다.

4.1 설계 환경

시뮬레이션을 위한 시스템으로 개인용 PentiumⅢ 700㎒ 컴퓨터를 사용하였으며 운영체제로는 리눅스를 이용하였고, 시뮬레이터로는 NS-2(Network Simulator-2)를 사용하고 그 결과를 저장하기 위하여 Axum을 사용하였다.

시뮬레이션의 신뢰성을 높이기 위해 인터넷상에 공개된 trace[1]를 사용하여 제시된 trace에 대해 적중률과 바이트 적중률에서 우수함이 입증된 알고리즘과 본 논문에서 제시한 알고리즘을 비교평가 하였다.

시뮬레이션은 적중률과 바이트 적중률을 고려하여 정해진 교체 알고리즘을 캐쉬의 파일타입별 클래스에 각각 적용시켜 1주일마다 클래스 분할비를 갱신시키는 방법을 사용한다. 이때 웹 캐쉬의 크기는 trace 총량의 10%(새로운 개체에 의해 캐쉬

내의 어떤 개체도 제거되지 않을 최소한의 크기)로 설정했다.

4.2 trace 특징

시뮬레이션의 결과는 입력으로 사용된 트레이스에 의존한다. 본 시뮬레이션에 사용된 trace에 대한 통계를 표 1에 나타냈다. 이를 보면 웹 trace의 높은 다양성을 알 수 있다.

표 1에서 Graphic 타입의 파일이나 Text/html 타입의 파일들이 전체 요청의 90% 이상을 차지함을 알 수 있다. 또한 크기가 작은 Graphic 타입의 파일들이 약 46%, Text/html 타입의 파일들이 약 29%로 요청이 많은 만큼 전체 전송량의 대부분을 차지한다는 것을 알 수 있다. 그리고 Audio, Video 타입은 요청은 미비하지만 객체의 크기가 크기 때문에 전체 전송량의 20% 정도를 차지함을 알 수 있다.

표 1. trace 통계

File Type	요청횟수	요청비율	전송량(MB)	전송비율
Graphic	27,551	51.13%	298.17	46.26%
Text/html	23,374	43.38%	188.85	29.30%
Audio	134	0.25%	115.44	17.91%
Video	21	0.04%	23.07	3.58%
CGI	511	0.95%	0.39	0.06%
Unknown	2,290	4.25%	18.63	2.89%
Total	53,881	100%	644.55MB	100%

4.3 클래스별 교체 알고리즘

우선 공개된 trace를 이용하여 분할된 각 클래스의 교체 알고리즘을 찾기 위해 모든 클래스에 LRU, LFU, SIZE, LRFU 교체 정책을 각각 한번씩 적용한 후 그 결과를 관찰하였다. 즉, 매번 동일한 trace를 사용하고 매번 다른 교체 알고리즘을 모든 클래스에 적용하여, 특정 클래스에서 좋은 성능을 보이는 교체 알고리즘들을 찾아내고자 했다.

표 2는 분할된 클래스에 각각의 교체 알고리즘이 적용되었을 때의 각 분할 클래스에서의 교체 알고리즘의 적중률(hit rate)을 보여주고, 표 3에서는 바이트 적중률(byte hit rate)을 보여주고 있다.

표 2. 각 교체 알고리즘의 적중률

	LRU	LFU	SIZE	LRFU
Graphic	41.3%	43.5%	45.1%	47.6%
Text	35.7%	38.2%	40.9%	42.7%
Etc	29.7%	30.4%	33.6%	32.5%

표 3. 각 교체 알고리즘의 바이트 적중률

	LRU	LFU	SIZE	LRFU
Graphic	42.2%	43.9%	40.5%	45.3%
Text	36.1%	37.7%	33.2%	39.8%
Etc	27.8%	28.3%	28.9%	28.5%

이 결과를 바탕으로 시뮬레이션에 사용될 각 클래스별 교체 알고리즘은 표 4와 같다.

표 4. 클래스별 교체 알고리즘

Class	교체 알고리즘
Graphic class	LRFU
Text class	LRFU
Etc class	SIZE

4.4 성능 평가 결과

일반적으로 웹 캐쉬의 성능평가 척도로는 적중률(hit rate), 바이트 적중률(byte hit rate), 사용자 응답시간, 그리고 전체 비용 등이 사용되고 있다. 인터넷 객체의 크기가 가변적인 성격을 가지기 때문에 본 논문에서는 성능평가 척도를 적중률과 바이트 적중률로 정한다.

그림 5와 그림 6은 시뮬레이션 결과를 보여준다. 그림 5는 날짜 흐름에 따른 적중률을 나타내고, 그림 6은 날짜 흐름에 따른 바이트 적중률을 보여준다.

그림 5와 그림 6에서 날짜 변화에 따른 적중률은 1~5% 정도의 성능향상이 있었고, 바이트 적중률은 2~9%의 성능향상이 있음을 알 수 있다.

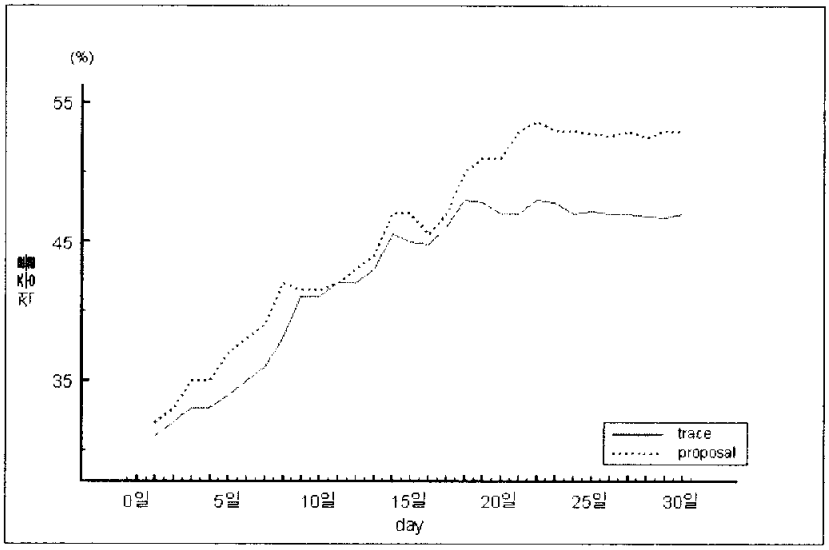

그림 5. 날짜 흐름에 따른 적중률

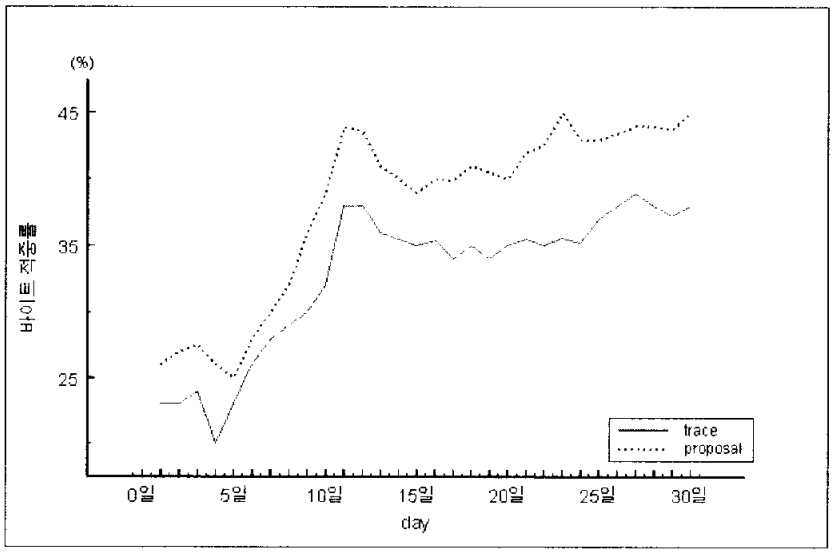

그림 6. 날짜 흐름에 따른 바이트 적중률

V. 결 론

최근 들어 웹을 기반으로 하는 인터넷 사용자의 급증으로 인해 서버와 망의 과부하 현상을 일으켜 사용자의 지연시간이 증가하고 있다. 이를 감소시키는데 일반적으로 사용되는 방법이 웹 캐쉬이다. 웹에서 요청되는 객체의 대부분이 이미지나 텍스트 파일 타입에 집중되고, 비디오나 오디오 파일타입 그리고 그 외의 파일 타입들은 상대적으로 요청이 작다. 이미지나 텍스트 파일 타입의 객체들은 그 크기가 작은 반면 비디오, 오디오는 이들의 수십배 내지 수백배에 해당하는 크기를 가지고 있다.

이처럼 웹에는 다양한 크기의 파일 타입이 존재하는데, 기존의 웹 캐쉬에서는 요청되는 객체의 가변적인 크기를 고려하지 않아 크기가 큰 객체가 크기가 작은 여러 개의 객체를 교체하여 캐쉬 적중률을 감소시키거나, 크기가 크고 빈도수가 적은 파일이 계속적으로 교체되어 바이트 적중률이 감소되는 문제점이 있었다.

이를 해결하기 위한 캐쉬공간을 특정 비율로 고정시켜 분할하는 방법은 분할된 각 영역에 같은 교체 알고리즘을 적용함으로써 분할된 캐쉬의 장점을 다 활용하지 못한다는 단점이 있다.

본 논문에서는 캐쉬를 분할했지만 각 영역에 같은 교체 알고리즘을 적용함으로서 발생하는 단점을 극복하기 위하여, 캐쉬 공간을 객체의 파일 타입에 따라 Graphic, Text, Etc 3부분의 클래스로 분할하고 각각의 클래스에 LRFU, LRFU, SIZE 교체 알고리즘을 적용하며, 웹 사용자들의 동적인 객체 요구 변화에 대응하기 위하여 1주일 단위로 분할비를 갱신시키는 알고리즘을 제시하여 성능이 향상됨을 알 수 있었다.

향후 연구로는 좀 더 다양한 트레이스 자료에 대한 성능분석이 이루어져야 할 것이다. 또한 통합된 캐쉬를 보다 효율적인 파일 타입별 클래스로 분할하는 방법과 캐쉬 분할비 갱신을 위한 최적의 주기를 도출하는 것이다.

참고 문헌

[1] S. Williams, M. Abrams, C. R. Standrige, G. Abdulla, and E. A. Fox, "Removal policies in network caches for world-wide web document", In Processings of ACM SIGCOMM '96, pp. 293-304, Aug. 1996.

[2] V. Almeida, A, Bestavros, M. Crovella, and A. Oliveira, "Characterizing reference locality in the WWW", In Proc. of the 4th Int'l Conf. on Parallel and Distributed Information Systems, pp. 92-103, 1996.

[3] S. Williams, M. Abrams, C. R. Standrige, G. Abdulla, and E. A. Fox, "Caching proxies : Limitations and potentials", Proc. of 4th International WWW conference, pp. 119-133, Dec. 1995.

[4] M. F. Arlitt and C. L. Williamson "Web server workload characterization: The search for invariants", In Proc. SIGMETRICS, Philadelphia, PA, pp. 126-137, Apr. 1996. ACM

[5] Cristina Duarte Murta, Virgilio Almeida, and Wagner Meira, Jr. "Analyzing performance of partitioned caches for the WWW". In Proceeding of the 3rd International WWW Caching Workshop, June, 1998.

[6] Jia Wang. "A survey of web caching schemes for the Internet". AGM Computer Communication Review, pp. 36-46, October, 1999.

[7] Pei Cao and Sandy Irani. "Cost-aware WWW proxy caching algorithms". In Processings of USENIX Symposium on Internet Technology and Systems, pp. 193-206, December, 1997.

[8] D. Lee, J. Choi, J. Kim, S. Noh S. Min Y. Cho, and C. Kim, "LRFU replacement policy: a spectrum of block replacement policies," In IEEE Transactions on Computers, pp. 1352-1361, 1996.

[9] Roland Peter Wooster, "Optimizing Response Time Rather than Hit Rate of WWW Proxy Caches", Master Thesis, Virgina Polytechnic Institute and State University, Dec. 1996.

[10] J. C. Bolot and P. Hoschka, "Performance engineering of the World-Wide Web : Application to dimensioning and cache design", Proc. of the 5th Int'l WWW Conf., pp. 1397-1405, 1996.

[11] D. Wessels and K. Claffy, Application of Internet Cache Protocol(ICP), version2, Internet Draft : draft-wessels- icp-v2-appl-00, Internet Engineering Task Force, 1997.

[12] D. A. Patterson and J. L. Hennessy. Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers, pp. 635-755, 1996.

[13] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K Page Replacement Algorithm For Database Disk Buffering", in Proceedings of the 1993 ACM SIGMOD Conference, pp. 297-306, 1993.

[14] A. Bestavros, R. L. Carter, M. E. Crovella, C. R. Cunha, A. Heddays, S. A. Mirdad, "Application- Level Document Caching in the Internet", Proc. of SDNE '95 2nd International Workshop on Services in Distributed and Networked Environments, Jun. pp. 166-173, 1995.

[15] 임재현, "프락시 서버 성능 향상을 위한 파일 타입 기반 캐싱 기법", 정보과학회 논문지(A), 제25권, 제8호, pp. 877-886, 1998.

[16] 최수영, 문진용, "인터넷 서버 부하 경감을 위한 효율적인 캐쉬 알고리즘", 한국정보처리학회 논문지, 제8-C권, 제 2 호, pp. 128-133, 2001.

감사의 글

길고도 짧았던 2년 반의 대학원 생활은 저에게 있어 소중한 것들을 많이 만든 귀한 시간이었던 것 같습니다. 많은 사랑과 관심으로 부족한 제가 더욱 성장할 수 있도록 도움을 주신 많은 분들께 진심으로 감사의 말씀 드립니다.

먼저 늘 걱정만 끼쳐드린 저에게 멀리서나 가까이에서 항상 따뜻한 말씀과 용기를 주시고, 즐거운 일이 있을 때 같이 기뻐해 주셨던 지도교수님 윤성대 교수님께 진심으로 감사의 말씀 드립니다. 바쁘신 와중에도 논문 심사를 위해 많은 관심과 조언을 아끼지 않으셨던 여정모 교수님, 많은 격려와 따뜻함을 보여주신 박홍복 교수님 감사합니다. 학부부터 석사과정까지 많은 가르침을 주신 박지환 교수님, 박만곤 교수님, 김영봉 교수님, 정순호 교수님, 박승섭 교수님, 이경현 교수님, 김창수 교수님께도 감사드립니다. 그리고 부족한 선배에게 많은 도움을 준 이미나 조교님께도 감사의 마음 전합니다.

늘 따뜻하고 인자하신 김용덕 교수님, 항상 밝고 즐겁게 대해주신 고석범 선생님, 항상 자기 일처럼 도와주고 지켜봐 주신 순환선배, 힘들 때 많은 도움과 격려를 아끼지 않으신 상일씨, 항상 도움이 못 되었다고 미안해하시지만 옆자리에서 정말 이것저것 많이 도와주신 박현호 선생님, 조용히 도움의 말을 많이 주신 박월선 선생님, 대학원 동기지만 많이 챙겨드리지

못해 항상 죄송하게 생각하는 송동영 선생님 정말 감사드립니다. 그리고 학회 논문준비 같이 하면서 이것저것 많이 챙겨주고 항상 즐거움을 주는 진현씨, 연구실의 귀여운 막내들인 봉현이와 문한이, 입학부터 지금까지 항상 도와주고 신경 써주시는 김영만 선생님, 배은호 선생님, 미소가 아름다운 순현 선배, 이쁜 도위 선배와 미란 선배, 귀엽고 성실한 후배 지영, 정애, 귀녀 그리고 연구실의 모든 식구들에게 감사드립니다.

힘들 때마다 같이 고민해주고 많은 충고와 사랑을 아끼지 않았던 형구, 정수, 태훈, 후숙이 언니, 안나와 대학원에서 같이 웃으면서 생활을 한 선배, 후배, 동기들께도 감사의 마음을 전합니다.

그리고 대학원과 직장 때문에 바쁘다는 이유로 부모님께 늘 소홀했던 딸에게 사랑으로 대해주신 아버지와 늘 편한 친구처럼 대해주시는 어머니께 정말로 감사드립니다. 끝으로 힘들 때 같이 고민해주고 위로해주며, 지금까지 불평한마디 없이 투정을 다 받아준 사랑하는 지웅 선배에게도 감사하다는 말을 전합니다.