캐쉬의 역할 구분을 이용한 확장성이 있는 캐쉬 그룹 구성 정책

A Scalable Cache Group Configuration Policy using Role-Partitioned Cache

현진일
명세정보시스템 기술연구소
민준식
경동대학교 정보통신공학부

중심어 : 캐쉬, 인터넷, 확장성, 프록시

요 약

오늘날, 인터넷의 급격한 증가에 있어서 응답지연과 네트워크 트래픽의 양, 그리고 서버의 부하를 줄이기 위한 파일 캐싱은 매우 중요해졌다. 실제로 하나의 네트워크에서 캐쉬를 사용함으로서 트래픽을 줄일 수 있게 되었고, 이것은 파일 캐싱이 인터넷 링크의 용량을 늘리기 위한 비용 부분을 개선할 수 있음을 의미한다. 본 논문에서 확장성 문제를 해결하기 위하여 동적 캐쉬 그룹 구성 정책을 소개한다. 모의 시험 결과는 본 논문에서 제안한 정책을 사용하는 캐쉬 그룹이 응답지연시간이 줄어들었음을 보여주며, 우리의 캐쉬 그룹 구성이 정적 캐쉬 구성보다 더욱 확장성이 있음을 보여 준다.

Jin-il Hyun (littleh@baubau.com)
Myung Se Information System ltd
Jun-Sik Min (jsmin@k1.ac.kr)
Div. of Information & Communition Eng.,
Kyungdong University.

Keyword : Cache, Internet, Scalability, Proxy

Abstract

Today, in exponential growth of internet, the importance of file caching which could reduce the server load, the volume of network traffic, and the latency of response has emerged. Actually, in one network, the traffic has reduced by using the cache and this means that file caching can improve the internet environment by cost a fraction of link upgrades. In this paper, we address a dynamic cache group configuration policy, to solve the scalable problem. The simulation result shows that the cache group using our proposal policy reduces the latency of response time and it means that our cache group configuration is more scalable than the static cache configuration.

I. 서론

인터넷 환경이 급속히 확장하는 가운데 서버의 부하 및 네트워크내의 통신량의 감소, 반응 지연 시간의 감소 등의 목적으로 캐쉬의 중요성이 대두되고 있다. 실제로 캐쉬를 채용한 한 네트워크 상에서 전체 통신량의 20%가량이 감소[1]한 것을 볼 수 있는데 이는 통신 선로를 개량하는 비용의 일부로 큰 성능 향상을 가져 올 수 있다는 것을 보여주고 있다. 이러한 캐쉬 운영 정책으로 현재 harvest, squid 방식이 가장 널리 사용되고 있으며 이를 응용, 변형한 몇몇 방법들이 사용되고 있는 상황이다. 그러나 위에서 제시한 캐쉬 구조들은 정적인 형태로 구성되어 있는데 이러한 구조로는 급속히 확장하는 캐쉬를 감당하는 데에 한계가 있다.

1. 연구배경 및 목적

정적 캐쉬 구성상에서는 고정적 경로를 따르는 데이터 흐름으로 인해 순간적인 요청 몰림 현상(hot spot)[2]에 대한 대처가 어려울 뿐 아니라 구조 유지 시 필요한 정보 교환으로 인해 확장시 전체적인 통신량을 증가시켜 확장성을 떨어뜨리는 결과를 초래한다. 이에 동적인 캐쉬 구조를 요구하게 된다. 본 논문에서는 이러한 동적 캐쉬 구조에 대한 모색으로 새로운 캐쉬 구성 정책을 제시하려 하는데 그 방법이 역할 캐쉬를 이용한 중첩 그룹 캐쉬 구조이다. 이 구조상에서 캐쉬는 두 가지 자격을 갖는데 그룹 내 캐쉬(ingroup cache)와 중첩 캐쉬(overlap cache)가 그것이다. 그룹 내 캐쉬는 기존의 캐쉬 역할만을 담당하게 되고 중첩 캐쉬는 기존 캐쉬 역할 뿐 아니라 그룹과 그룹의 연결 역할의 수행과 그와 관련

접수번호 : #030805-001
접수일자 : 2003년 8월 5일, 심사완료일 : 2003년 9월 23일

*교신저자 : 민준식, e-mail : jsmin@ki.ac.kr

된 효율적인 운영 기술들을 포함하게 된다.

II. 기존연구

초기 인터넷 캐쉬는 서버와 클라이언트 사이의 proxy 캐쉬로써 독점적 역할을 담당하였다. 인터넷 환경의 확장은 이들 캐쉬 사이의 공유를 요구하게 되었고 Harvest project는 ICP(Internet Cache Protocol)를 통한 주변 캐쉬들과 내용을 공유하는 방법을 제시하여 처음으로 "Web cache sharing" 개념을 도입하였다[3]. 그러나 ICP를 이용한 캐쉬 사이의 공유는 캐쉬 수가 증가할수록 통신량의 증가와 반응 지연시간(latency)의 증가를 가져와 확장성에서의 한계를 드러내게 된다. 본 장에서는 캐쉬의 공유 방법을 제시하는 기존 연구들을 살펴보고 이들 방법들의 문제점과 해결책을 모색한다.

1. 기존 캐쉬 구성 정책

1.1. 프록시 캐쉬

프록시 캐쉬는 네트워크상에서 웹 통신량을 줄이는 방법으로 오래 동안 사용되어진 정책이다[1]. 프록시를 사용할 경우 클라이언트의 페이지 요청은 처음에 서버까지 직접 전달되어 지는 것이 아니라 프록시에 전달되게 된다.

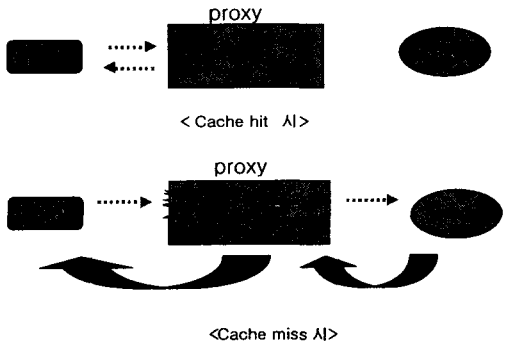


그림 1. 프록시 캐쉬

이 경우 해당 페이지가 프록시에 존재할 경우 프록시에서 요청에 대한 반응이 이루어지고 존재하지 않을 경우 프록시는 해당 페이지를 서버에 요청하게 된다. 다시 말해 이 경우 서버로의 요청의 주체는 클라이언트가 아닌 프록시가 되게 된다. 해당 페이지를 서버로부터 받아온 프록시는 자신의 공간에 페이지를 복사하게 되고 다음 요청부터는 반응이 프록시에서 가능하게 된다. 이 경우 서버는 각 개별 클라이언트로부터의 요청을 프록시 하나로부터 받음으로써 서버에 걸리는 부하를 줄이고 클라이언트는 요청이 네트워크를 지나는 지연을 줄임으로써 전체적인 웹 성능을 향상시키는 중요한 정책이다. 그러나 각 프록시에 연결된 클라이언트가 증가하면 자신 스스로가 요청으로부터 처리 한계에 부딪치게 된다.

1.2. 공유 캐쉬

Malpani[4]는 그의 논문에서 캐쉬의 그룹을 형성하기 위한 시도를 보여주었다. 프록시 캐쉬 정책과 마찬가지로 클라이언트는 자신이 접속하는 캐쉬에서 페이지를 탐색하고 여기서 적중이 일어날 경우 반응하게 된다. 그러나 접속 캐쉬에서 적중이 일어나지 않을 경우에는 프록시 방법과 다르게 서버가 아닌 주변의 캐쉬들을 먼저 탐색하게 된다. 이를 위해서는 주변 그룹과의 통신 방법을 모색해야 하는데 가장 일반적인 방식이 요청 멀티캐스트 방식이다. 이를 위해서는 자신이 탐색할 주변 캐쉬들을 미리 정해 놓아야 하는데 이로써 그룹의 형성이 필요하게 된다. 이런 방법을 통해 주변 캐쉬의 자원을 공유하여 활용할 수 있다. 그러나 자신이 속한 그룹안에서 해결되지 않은 요청은 주변에 다른 캐쉬가 존재하더라도 다른 그룹의 캐쉬를 이용할 수 없다. 이로써 주변 캐쉬를 보다 유용하게 이용하기 위해서는 캐쉬의 그룹을 확대할 수 밖에 없지만 그리하게 되면 그룹내 통신량이 많아져 통신지연이 생기게 된다. 즉 이 방식 또한 확장성의 한계를 가지게 된다.

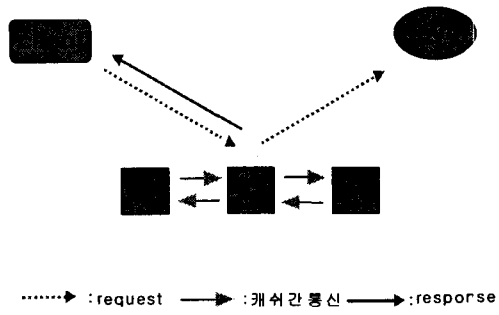


그림 2. 공유 캐쉬

1.3. 계층 캐쉬

계층 캐쉬 구조는 클라이언트가 접속한 캐쉬에서 미스가 이루어 질 경우 같은 수준의 캐쉬들을 ICP를 통해 검색하고 여기서도 적중이 이루어지지 않을 경우 상위 수준(parent cache)에 다시 요청하는 방향으로 요청 진행이 이루어지는 방식이다[5]. 그러나 이러한 구조는 상위 수준의 캐쉬로 갈수

록 병목 현상을 유발하여 순간적 요청에 대해 반응성이 떨어지는 결과를 가져오며 계층의 깊이가 깊어질수록 하위 수준의 형제 노드들이 증가하여 ICP를 통한 페이지 탐색이 길어져 반응 지연 시간의 증가를 초래하게 된다.

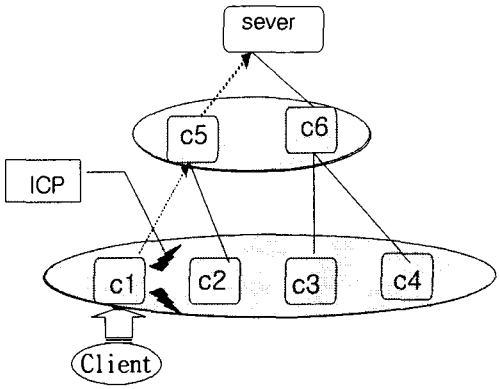


그림 3. 계층 캐쉬

2. 적응성 계층 캐쉬

계층 캐쉬의 병목 현상 문제는 요청이 빈번한 페이지를 계층의 하부 즉 클라이언트에 가까운 위치에 두어 요청 흐름이 상위 수준까지 올라가는 것을 막음으로써 해결이 가능하다. 이를 위해서 요청량을 기준으로 해당 페이지를 하부 캐쉬에 복사해 나가게 되는데 이러한 방식을 적응성 계층 구조라 한다[6]. 적응성 캐쉬 구조에서는 일정 기간 동안의 요청량이 어느 기준치를 넘으면 하위 수준의 캐쉬에 페이지 복사가 이루어지고 다음 흐름의 요청 반응은 복사가 이루어진 캐쉬, 즉 이전보다 하위 수준의 캐쉬에서 담당하게 된다. 여기서도

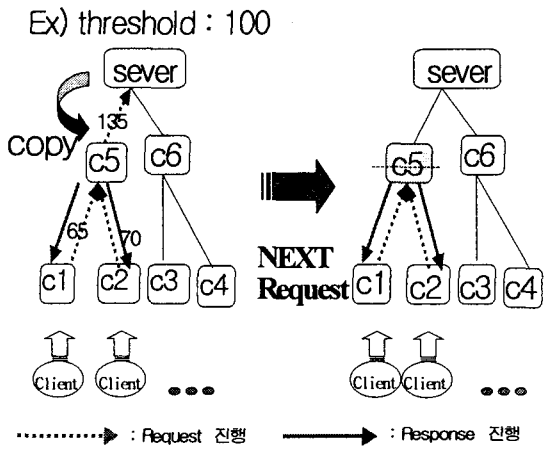


그림 4. 적응성 계층 캐쉬

요청량이 임계값(threshold)을 넘으면 보다 하위 수준의 캐쉬에 페이지 복사가 이루어지고 이 과정을 반복하여 클라이언트에 가까운 위치에 해당 페이지를 위치시키게 되는 것이다.

그러나 이 방법 역시 서버로의 순간적인 몰림 현상은 제어할 수 있지만 꾸준한 일반적 빈도수(임계값을 넘지 않는)의 요청에서는 서버의 부담을 줄이는데 한계가 있다.

그림 4를 보면 적응성 반응이 어떠한 방식으로 적용되는지 그 모습을 그래프로 나타내었다. 원래 페이지를 가지고 있는 서버는 초기 클라이언트로부터 요청을 받게되는데 그때에서는 해당 요청은 C1,C2,C3,C4 캐쉬로부터 이루어진다. 이때 서버에 이르는 요청은 C1,C2 그리고 C3,C4가 서로 다른 경로로 올라오는 데 각 경로의 요청량을 각각 계산하여 서버가 해당 페이지를 자신보다 한 수준 아래의 캐쉬에 저장할 것인지를 결정하게 된다. 여기서 서버는 자신보다 한 수준 아래 캐쉬들에 대하여 각각 기준치 값을 적용하게 되는데 기준치가 100일 경우 C1,C2를 타고 올라와 C5를 거치는 요청은 이를 넘으므로 서버는 한 수준 아래의 C5 캐쉬에 페이지 복사를 이루게 된다. 이럼으로써 다음 요청은 서버까지 올라가지 않고 C5에서 처리됨으로 서버는 C1,C2로부터 올라오는 요청 메시지로 인한 서버 부담과 서버와 C5의 네트워크 자원을 절약할 수 있다. 이 논문에서는 이러한 적응성 캐쉬의 장점을 그대로 차용하였다.

III. 중첩 캐쉬 구조의 응용

1. 중첩 캐쉬의 응용

중첩 캐쉬가 가져야할 구조는 다음과 같다. 먼저 역할 캐쉬는 그룹의 캐쉬를 두 가지 기능적 유형으로 나눈 것으로 그룹내 캐쉬와 중첩캐쉬로 분류되며 캐쉬 그룹 식별자에 의해 구분된다.

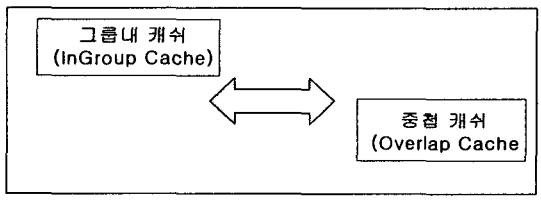


그림 5. 역할 캐쉬 구분

그룹내 캐쉬는 일반적인 캐쉬가 수행하는 역할을 하게 된다. 이 캐쉬는 페이지에 대한 저장 역할을 수행하고 클라이

언트가 직접 접속하는 1차적 접근 노드로 사용되기도 한다.

중첩 캐쉬의 경우는 그룹내 캐쉬의 모든 기능을 수행할 뿐 아니라 그룹 연결자, 요청 진행 결정자, 1차적 적응성 캐싱 등의 역할을 한다. 이를 위해서 먼저 중첩 캐쉬는 여러개의 그룹 ID 목록을 유지하게 된다. 또한 요청 진행 테이블을 가지고 있어 진행되어온 요청을 해당 테이블을 바탕으로 진행 방향을 결정하게 된다. 초기 페이지 저장에서 캐싱에 사용되어 진다.

다음은 group ID의 역할이다. 중첩된 캐쉬 구성 정책이 기존 계층 구조 캐쉬와 구별되는 가장 큰 특징은 캐쉬가 서로 다른 여러 그룹에 공통적으로 속하게 되어 그룹과 그룹의 연결 역할을 한다는 것이다. 이렇기 때문에 자신이 속한 각각의 그룹을 구별할 수 있어야 하고 각 그룹의 정보를 효율적으로 저장하는 방법이 필요하다. 이를 위해 group ID라는 개념을 도입한다. group ID는 그룹 내 캐쉬에서는 자신이 속한 그룹을 확인하는 요소로 사용되고 요청 메시지를 멀티캐스트 할 때는 메시지가 도달하는 범위 한정 역할을 한다. 다시 말하면 그룹의 각 캐쉬는 그룹 유지에 필요한 정보 교환시 자신이 속한 그룹을 확인할 때 group ID를 통해 인지하고, 페이지 요청 진행시 멀티캐스트되는 요청은 각 캐쉬에 저장하고 있는 group ID를 비교하여 동일한 ID를 가진 캐쉬에만 전달됨으로써 동일 그룹내의 캐쉬만이 요청 메시지를 얻어낼 수 있다.

이러한 요소들을 종합해서 중첩 캐쉬 구조를 도식화한 것이 그림 6이다.

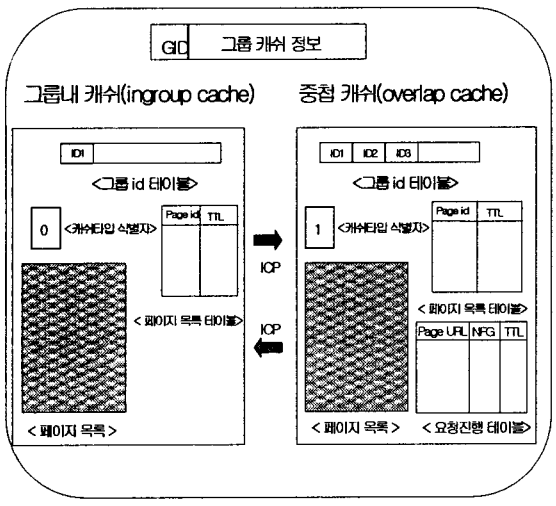


그림 6. 중첩 캐쉬의 그룹 구성

한편 group ID를 유지하기 위해서는 그룹의 변화를 중요하게 고려하여 적절한 정책을 제시한다.

그룹이 생성되거나 소멸 또는 합쳐질 경우 기존 그룹들의 ID는 변화가 필요하다. 이 경우 일반적으로 생각할 수 있는 것이 중앙 집중적인 방식으로 group ID를 관리하는 서버를 통해 그룹 생성 시는 새로운 ID를 부여하고 소멸 시는 해당 ID를 환수하며 결합 시는 새로운 ID를 부여하면 된다. 그러나 이러한 방식의 경우 관리 서버를 새로 두어야 하고 이 서버는 모든 그룹의 정보를 항상 유지하여야 한다. 따라서 본 논문에서는 각 그룹들이 주기적으로 자신의 그룹 상황을 교환하는 방식을 채택하여 그룹의 변화에 적절한 ID 운영정책을 수립한다. 이 경우 그룹간에 정보 유지를 위해 발생하는 통신량 증가는 페이지 요청 시 해당 정보를 piggyback 형식으로 처리하여 최소화한다. 즉 그룹 생성 시는 주변 정보를 통해 사용되지 않는 ID 하나를 생성하고 소멸 시는 자신의 ID 소멸을 주변 그룹에게 알려주면 된다. 한편 결합 시는 두 ID를 모두 소멸하고 새로운 ID를 공통으로 부여받는 방법을 고려할 수 있으나 이 경우 두 그룹의 모든 캐쉬의 group ID를 변경하여야 한다. 이 경우 두 그룹의 크기(ex: 캐쉬 수)를 비교하여 큰 그룹의 ID를 승계한다면 이에 드는 비용을 절반 이상을 줄일 수 있다. 그림 7은 이러한 예를 나타낸 것이다.

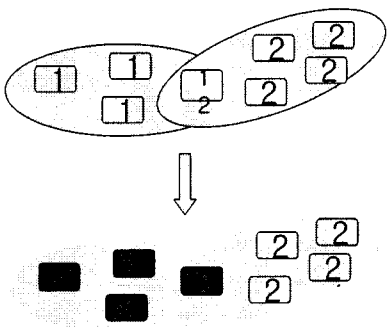


그림 7. 그룹 합병시 group id 변화

2. 캐쉬 운영 정책

다음은 요청 메시지 및 데이터(페이지) 진행 상황을 바탕으로 중첩 캐쉬가 어떻게 운영되는지 나타낸 것이다.

요청(request) 측면에서 바라본 흐름을 살펴보면, 클라이언트에 의해 들어온 요청은 일단 자신의 지역 캐쉬를 탐색하게 된다. 여기서 미스가 일어나면 다음으로 그 캐쉬가 속한 그룹의 다른 캐쉬들을 찾게 되는데 이때 두 가지 방법을 생각

할 수 있다. 멀티캐스팅 방식과 해싱 테이블을 이용한 탐색 방식이다. 해싱 테이블을 이용하는 방식은 각 캐쉬가 자신이 속한 그룹의 다른 캐쉬들이 저장하고 있는 페이지 정보를 유지함으로서 이를 통해 요청한 페이지가 있는 캐쉬를 바로 찾아가는 방식이다. 이러한 방식을 채택할 경우 중첩 캐쉬는 여러 그룹의 캐쉬 목록 테이블을 유지하여야 하는 문제가 발생하게 된다. 따라서 여기서는 멀티캐스팅 방식을 사용한다. 이 경우 그룹 내 통신이 많아지는 단점이 있지만 그룹 사이즈가 작을 경우 그룹 내 통신량이 그리 크지 않게 되며 각 캐쉬가 지역 내 다른 캐쉬들의 정보를 유지하여야 하는데 드는 비용도 없게 된다. 처음 단계 즉 자신이 접속한 캐쉬가 속한 그룹에서 멀티캐스트 방식의 요청에 의한 적중(hit)이 일어나지 않을 경우 요청은 기본적으로 해당 그룹의 중첩 캐쉬들에 보내지게 된다. 여기서 다른 그룹으로의 요청 전이가 일어나게 된다. 다시말해 중첩 캐쉬는 다시 요청의 원천이 되어 다시 해당 그룹에 멀티캐스트 요청이 이루어진다. 이러한 과정을 거듭하여 각 그룹에서 계속해서 캐쉬 미스가 일어나면 최종적으로 서버에서 반응(response)이 이루어지게 된다.

다음으로 반응 측면에서의 흐름을 나타낸 것이다. 서버까지 올라온 요청은 서버에서 적중이 일어나고 반응이 이루어지게 된다. 한편 적응성 캐쉬 구조를 형성하기 위해서는 반응이 이루어진 페이지를 단지 클라이언트에 전달하는 것에서 그칠 것이 아니라 하부 수준 캐쉬에도 페이지 저장이 이루어져야 한다. 기존에 발표된 논문[6]에서는 반응이 일어난 그룹의 모든 캐쉬에 반응 페이지가 멀티캐스팅 방식으로 전달되게 되고 반응 그룹 내 모든 캐쉬들이 해당 페이지를 일정 기간동안 저장하는 정책을 취하도록 하고 있다. 이렇게 해서 다음 흐름의 요청은 서버까지 올라가지 않고 그 바로 전 단계에서 적중이 일어나게 된다. 그러나 이러한 방식에서는 그룹 내 주변 캐쉬에 너무 많은 저장 장소를 요구하게 된다. 그림 8이 기존 방식인 그룹내 모든 캐쉬에 대한 멀티캐스트 복사가 이루어진 경우이다.

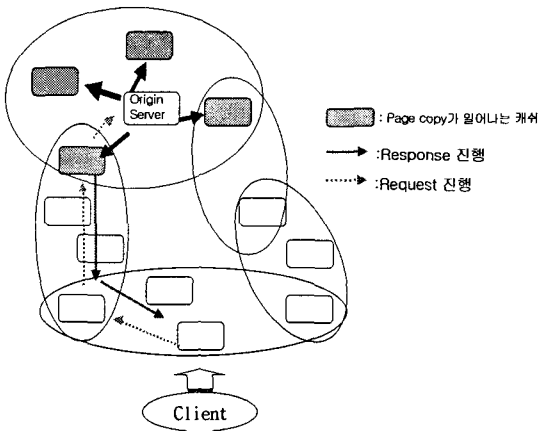


그림 8. 기존 방식을 이용한 중첩 캐쉬

따라서 본 논문에서는 적중이 일어난 그룹내의 모든 캐쉬에 반응 페이지가 멀티캐스팅되는 방식이 아닌 해당 그룹의 중첩 캐쉬에만 멀티 캐스팅 되는 방식을 택한다.

여기서 제기될 수 있는 문제가 경우에 따른 1hop의 요청 진행의 오버헤드와 그룹 내 캐쉬의 기능 상실 문제이다.

1hop의 요청 진행의 오버헤드 문제는 요청이 서버가 포함되는 그룹의 그룹내 캐쉬에 처음으로 접속하는 경우에 발생하는데 이 경우 그룹내 모든 캐쉬가 해당 페이지를 가지고 있는 경우 바로 반응되지만(그림 9의 경우) 본 논문에서 주장하는 중첩 캐쉬에만 적응성 있게 복사될 경우 그룹 내 캐쉬에서 바로 적중이 일어나지 못하고 한 hop을 더 거쳐 서버에서 반응(그림 10의 경우)하게 된다는 것에서 문제가 된다.

그러나 이것은 한번의 미스(1 hop의 overhead)이고 반응시의 통신량을 줄임으로써 보다 효율적인 공간 유지와 통신 복잡성을 줄일 수 있다. 이는 요청시의 패킷은 요구 시 필요한 정보(원하는 페이지 정보)만을 가지고 있지만 반응 시에는 해당 페이지 자체를 이동시켜야 하므로 반응시의 데이터 이동량이 요청시보다 훨씬 많다는 점에서 기인한다.

다음은 기존 방식의 그룹 구성과 본 논문이 주장하는 그룹 구성 방식에서의 데이터 흐름을 나타낸 것이다.

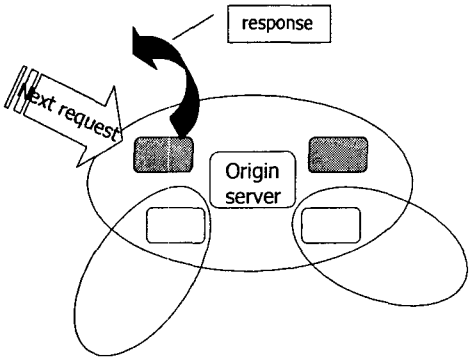


그림 9. 기존 그룹 구성상의 데이터 흐름

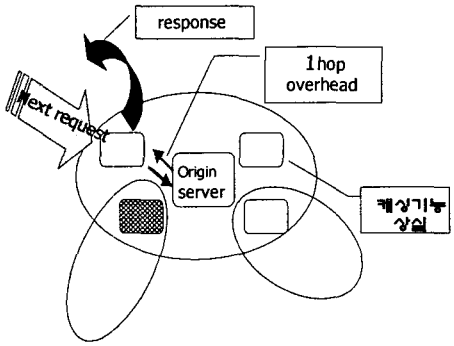


그림 10. 역할 캐쉬를 이용한 그룹 구성상의 데이터 흐름

두 번째 문제는 그룹 내 캐쉬들이 캐쉬 기능 상실의 문제이다. 반응된 페이지가 중첩 캐쉬에만 저장된다면 그룹 내 다른 캐쉬들은 캐쉬 기능을 발휘할 수 없게 된다. 그래서 적중이 일어난 그룹의 중첩 캐쉬 뿐 아니라 클라이언트가 접속한 캐쉬에도 해당 페이지를 내용을 저장한다. 클라이언트가 접속한 캐쉬는 중첩 캐쉬 뿐 아니라 그룹내의 모든 캐쉬일 수 있는 것이기에 이렇게 함으로써 그룹 내 캐쉬도 캐쉬 기능을 수행할 수 있다. 이를 구현할 수 있는 방법으로 각 요청 흐름마다 계수기(counter)를 달아 count[hit](hit는 적중 일어난 그룹까지의 총 그룹의 경로 수)인 곳에서는 중첩 캐쉬에 멀티캐스트를 통한 캐싱이 이루어지고 count[0] (client의 요구를 받은 캐쉬)인 곳에서는 일반적인 캐싱이 이루어지게 하는 것이다.

이러한 모든 것들을 종합해서 중첩 캐쉬가 운용되는 모습을 보면 그림 11과 같다.

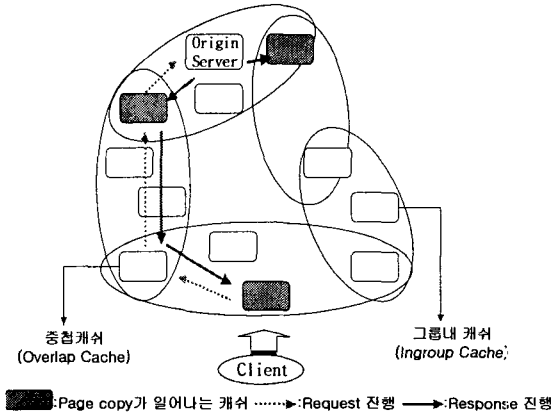


그림 11. 역할 캐쉬를 이용한 중첩 캐쉬 구조

다음으로 캐쉬내의 페이지 운영 정책을 살펴보면, 적응성 캐쉬를 구성하기 위해서는 각 캐쉬에 캐싱되는 페이지들을 어떻게 유지하여야 하는 문제도 고려하여야 하는데 여기서는 가장 일반적이고 효과적인 TTL(Time to Live) 방식을 사용한다. 각 캐쉬는 자신이 포함한 페이지에 대한 목록을 유지하게 된다. 이 목록에는 해당 페이지의 크기, 페이지가 저장된 시간, TTL 값 등 해당 페이지에 정보를 가지게 된다. 이때 이 TTL 값은 해당 페이지가 유효한 기간을 명시함으로서 분산된 페이지의 일관성 유지 등에 사용된다. 즉 각 페이지가 저장될 때 해당 페이지는 유효한 시간이 정해져 있으며 이 기간이 지나면 해당 페이지는 순수성이 훼손되어 서버에 다시 최신의 페이지를 요구하게 된다. 여기서는 이러한 일관성 유지를 위한 TTL 값이 아닌 캐쉬의 페이지 저장 제어를 위한 TTL 기술을 제시한다.

적응성 계층 캐쉬에서 페이지가 하부 노드로 복사되어 저장되는 기준은 해당 페이지에 대한 요청 빈도였으며 이는 일정 시간 동안 임계값을 넘는 요청을 받는 페이지에 대해서 복사를 진행함으로써 가능하였다. 즉 여기서는 이러한 조건을 만족하기 전에는 페이지 복사가 이루어지지 않는다. 그러나 본 논문에서 제안하는 방법을 사용할 경우에는 처음 요청부터 페이지 복사가 이루어진다. 따라서 기존 방식처럼 복사할 페이지를 결정하는 방법이 아닌 복사된 페이지 중 제거해야 할 페이지를 결정함으로써 캐쉬의 저장 장소가 제어된다. 이때 제거될 페이지를 결정하는 방법에 TTL 기술을 적용한다. 초기 TTL 값은 해당 그룹의 총 TTL 값의 합으로 하고 해당 페이지에 대해 요청이 오는 비율, 즉 총 요청중 해당 페이지가 차지하는 비율에 따라 TTL 값을 조정해 가는 방식

을 적용한다. 이렇게 함으로써 요청이 많이 오는 페이지는 캐쉬에 더욱 오래 남아있게 하고 따라서 결과적으로 이런 페이지가 보다 빨리 각 그룹에 전달되게 함으로써 궁극적으로 인기 있는 페이지를 클라이언트에 가깝게 위치시킬 수 있는 것이다.

다음 그림 12는 TTL 값을 계산하는 식으로 캐쉬내 전체 요청량과 해당 페이지에 대한 요청 비율을 기본 TTL 값과 연관하여 나타낸 것이다. 이렇듯 캐쉬 내 상황을 TTL 값 결정에 활용함으로써 각 캐쉬마다 각자 자신의 상황에 맞는 적절한 TTL 값을 찾을 수 있다.

$$TTL = \frac{CPR}{CTR} \times TTL_{default}$$

CTR : 캐쉬 내 총 요청량(Incache Total Request)
CPR : 캐쉬 내 해당 페이지 요청량(InGroup Page Request)
TTL default : 캐쉬 내 TTL값의 총합

그림 12. 적응성 캐쉬를 위한 페이지 TTL 계산 식

3. Request forwarding

여러 그룹이 중첩되어 존재하다 보면 원하는 페이지를 찾아 들어가는 경로가 여럿이 존재할 수 있다. 이러할 경우 전체 그룹내에 다른 경로를 따라가는 불필요한 요청 메세지가 생겨날 수도 있고, 이는 적응성 반응 구조를 생성하는데 장애 요소로 작용할 수 있다. 본 논문에서는 이를 각 페이지 요청에 대한 최단 이동 경로의 방향 진행 테이블을 중첩 캐쉬에 구현함으로써 해결한다. 반응(response)되어 돌아오는 메시지는 자신이 거쳐간 경로를 그대로 따라오게 된다. 이때 각 중첩 캐쉬가 돌아오는 메시지의 바로 전 단계의 그룹을 기억하게 된다면 역으로 다음 번 진행 때 이 그룹을 요청 진행 방향 그룹으로 활용할 수 있을 것이다. 즉 방향 진행 테이블에는 해당 페이지의 바로 다음 진행 방향 그룹의 ID만을 유지하게 되는 것이고 이것만으로도 정확하고 경제적인 경로의 요청 진행이 이루어질 수 있다.

물론 전체 경로를 테이블에 유지하는 방향도 생각해 볼 수 있다. 이 경우 전체 그룹 개수가 1,000개라 하더라도 한 그룹 표시에 210, 즉 10비트의 저장장소와 한 요청에 대해 경로는 최악의 경우 10비트*1,000 만큼의 이동경로가 필요로 하게된다. 이는 각 요청 페이지당 2KB 미만으로써 캐쉬 저장 장소의 크기에서 감당할 수 있는 크기이다. 그러나 이런 식의 전체 경로를 가지고 있을 경우 다음 페이지 요청은 각 중첩 캐쉬를 거칠 때마다 해당 경로 중 현재 위치를 찾아야 하는 계산이 필요하고 해당 페이지의 경우 한 주기마다 고정적인 경로를 따라 움직여야 한다. 또한 검색 속도를 감안하여 진행 테이블이 디스크가 아닌 메모리에 저장된다는 점에서 저장 장소의 감소가 필요하다. 이 같은 점들을 고려할 때 바로 다음 경로만을 유지하고 있을 경우, 페이지 진행 테이블의 저장 장소 절약 뿐 아니라 요청 진행을 각 진행 그룹 단계마다 동적으로 바꾸어 줄 수 있어 유리하다. 즉 전체 경로 저장시에는 중간에 부분적인 경로 변경이 있을 경우 전체 경로를 모두 바꾸어 주어야 하나 다음 경로 그룹 아이디만을 유지할 경우는 중간에 부분적인 경로 변경이 있을 경우 해당 중첩 캐쉬내의 요청 진행 테이블의 Next Group ID만을 바꾸어 주기만 하면 됨으로 빈번한 경로 진행 변경이 가능하게 된다. 여기에 TTL[7] 개념을 도입, 일정 간격으로 페이지 진행 테이블을 갱신한다면 보다 효율적인 동적 요청 흐름 제어가 가능한 것이다.

각 그룹의 중첩 캐쉬는 해당 테이블에 다음과 같이 요청한 페이지를 식별할 수 있는 식별자, 그리고 이전 요청에 의해 해당 페이지가 나아간 경로 중 현 그룹의 바로 다음 경로 group ID, 그리고 해당 행이 인정될 수 있는 시간(TTL)을 가지고 있어야 한다.

식별자의 경우 일반적으로는 해당 페이지가 있는 서버의 URL 주소가 될 것이고 해당 URL 여러 페이지로 되어 있을 경우에는 페이지 ID가 포함될 것이다.

Next Group Id는 반응되어 돌아오는 패킷의 경로에서 현 그룹에 이르는 바로 이전 단계의 그룹 아이디를 말한다. 이 경우 요청에 대한 최단 이동 경로의 group ID를 테이블 형태로 저장함으로써 해결한다. 그림 13은 요청 진행 테이블의 예이며, 그림 14는 그룹내에서 요청 진행 테이블이 구현된 경우이다.

URL table (page id)	Next Group id	TTL
www.dgu..	12	6
www.snu..	14	4
www.cau..	7	7

그림 13. 요청 진행 테이블 예시

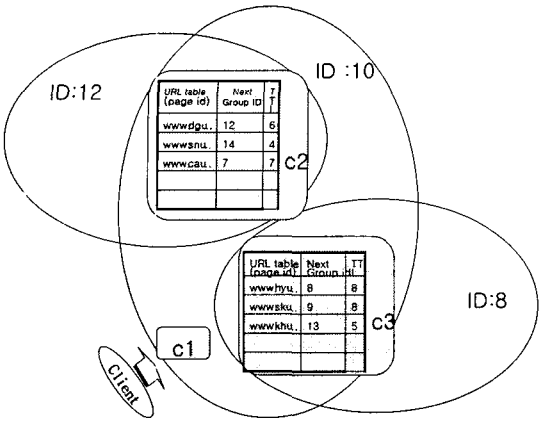


그림 14. 그룹내의 요청 진행 테이블 구현

다음은 TTL 목록에 관한 내용이다. 여기서는 3.2절의 캐쉬 목록을 제어하기 위한 TTL이 아닌 요청 테이블 유지를 위한 새로운 TTL 값을 제시한다. 이 TTL은 해당 행 정보가 얼마나 유효한지를 나타내 준다. 이는 동적 진행 방향으로 나아가는 가장 결정적 역할을 해주는 요소로 TTL 시간이 소멸되기 이전 페이지 요청은 현 테이블 정보를 바탕으로 진행되어지게 된다. 그러나 이 값이 소멸된 이후에는 해당 정보는 그 유효성을 잃고 해당 테이블에서 제거되게 된다. 따라서 다음 요청은 다시 경로를 찾게 되고 이때 최단 반응 시간을 주는 경로로 다시 요청 테이블에 추가되게 된다. 이때 각 중첩 그룹에 있는 해당 페이지의 TTL 값은 같을 필요가 없다. 다시 말해 각 중첩 캐쉬에서 동일 페이지에 대해 서로 다른 TTL 값을 가지게 되고 이는 효율적 요청 진행을 가능하게 해 준다.

한편 해당 그룹에서의 TTL 값은 해당 그룹의 통신량에 영향을 받아 결정된다. 통신량이 많은 그룹은 그만큼 통신 지연시간이 길어진다. 따라서 반응 지연 시간을 줄이기 위해서는 이러한 그룹을 피해 요청을 진행하는 것이 바람직하다. 이를 위해 페이지에 대한 경로가 저장될 때 해당 그룹의 통신량에 반비례하는 TTL 값을 설정한다. 이는 통신량이 몰리는 그룹은 빨리 요청 테이블을 변경시켜 주어 그룹내 통신량을 조절해 줄 수 있게 해준다. 이를 구하는 방식이 그림 15 이다.

$$TTL = \frac{ASGT}{SGT} \times TTL_{default}$$

SGT: The Size of Group traffic
ASGT: The Average Size of Group Traffic
TTL : average TTL

그림 15. 동적 경로 형성을 위한 TTL

따라서 동일 페이지에 대한 요청이라 하더라도 요청이 진행되어지면서 거치는 그룹의 상황에 따라 서로 다른 TTL 값을 갖게 되고 따라서 요청 진행 테이블의 TTL 값도 그룹마다 상이하게 된다.

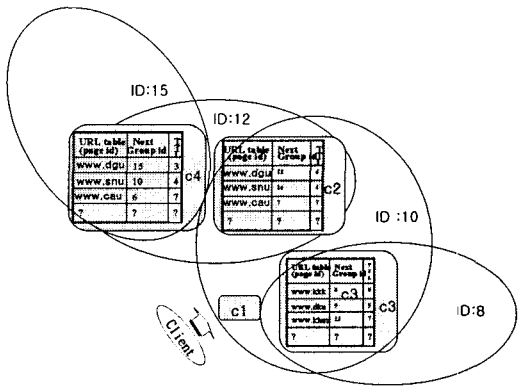


그림 16. 요청 진행 테이블내 TTL 값 분석

그림 16에서 볼 수 있듯이 동일 URL(또는 Page ID)을 가지고 있는 페이지에 대한 요청에서 각 중첩 캐쉬의 요청 테이블에 있는 TTL 값이 서로 다른 것을 볼 수 있다. 즉 www.dgu...에 대한 TTL 값이 그룹 12에서는 3, 그룹 10에서는 6임을 알 수 있다. 이는 그룹 12가 그룹 10에 비해 보다 많은 그룹 내 통신량을 가진다는 것을 의미하기도 한다.

IV. 구현 및 성능 평가

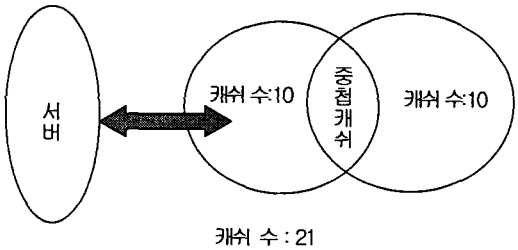
1. 구현환경

구현은 펜티엄 PC의 Windows 2000 운영체제상에서 이루어졌으며 시뮬레이터로써는 NS-2(Network Simulator-2)가 사용되었다. 본 논문의 목적이 중첩 캐쉬 구조를 이용한 동적인 확장성 캐쉬 구조를 이루는 것인 만큼 이를 보이기 위해 요청은 일정한 크기(초당 10개)로 제한하였으며 성능 평가의 초점은 전체 캐쉬수의 변화에 따른 반응 시간의 추이 변화에 맞추었다. 서버는 실제로 구현되고 있는 캐쉬 구조와 차이를 보이지만 두 실험 환경을 형평성을 위해 계층 캐쉬에서는 최

상위 캐쉬에 서버를 위치 시켰으며 중첩 캐쉬는 group ID1에 속한 그룹내 캐쉬에 서버를 연결하였다.

캐쉬는 실험의 편의를 위해 각 구성에 제한을 두었는데 계층 캐쉬의 경우 4-nary로 하였고 각각 3레벨, 4레벨, 5레벨로 구성하였다. 중첩 캐쉬의 경우 그룹내 캐쉬의 수를 정확한 수치 계산으로 적당한 캐쉬수를 구해야 하지만 여건상 그룹내 캐쉬수를 10으로 고정하였다. 따라서 비교할 캐쉬 수는 21, 85, 331개로 하였는데 이는 계층 캐쉬의 노드 수에 기인하였다. 즉 4-nary 3레벨의 계층을 생성할 경우 총 21개의 캐쉬로 4레벨인 경우 85개 5레벨의 경우 331개가 필요하기 때문에 중첩 캐쉬 역시 이들 캐쉬 수에 맞게 정하였다.

클라이언트의 경우 계층 캐쉬에서는 최하위 수준에 위치한 캐쉬에 각 1개의 클라이언트가 연결되도록 연결하였으며, 중첩 캐쉬의 경우 각 그룹의 그룹내 캐쉬에 각각 1 클라이언트를 연결하였다. 그리고 각 노드들의 연결은 1.5M TCP/IP 링크로 통일하였다.



캐쉬 수 : 21

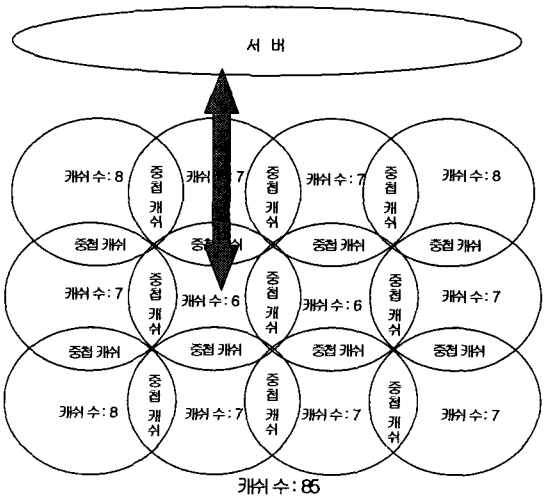


그림 17. 중첩 캐쉬의 구성 모습

시뮬레이션의 실행은 NS-2 시뮬레이터의 페이지 생성 옵션을 통해서 임의적인 trace를 생성하였다. 해당 trace는 (클라이언트 주소, 목적 페이지 주소, 요청 시간, 반응이 이루어진 시간)형식으로 생성되어지며 요청 페이지의 성격은 종합 페이지가(PagePool/Com-Math) 아닌 단일 페이지(PagePool/Math)로 하였으며 각 형성 페이지의 크기는 일반적인 페이지 크기인 100~10,000 바이트 사이로 제한하였다[8]. 기준값은 계층 캐쉬의 경우 5, 중첩 캐쉬의 경우 TTL 값을 1초로 하였다.

계층 캐쉬의 경우 NS2에서 제공하는 예제 중 계층 캐쉬 부분을 변경하였으며, 중첩 캐쉬의 경우 그림 17과 같은 layout으로 형성하였다.

2. 성능평가

실험 결과는 100초 동안에 1,000개, 1,000초 동안 10,000개의 요청을 임위적 간격으로 생성하여 한 페이지 요청당 요청 시간부터 반응이 이루어진 시간 사이의 차를 반응시간으로 하여 각 요청에 대해 요청별 반응시간을 평균을 내어 계산하였다(단위 시간은 초(sec)이다). 여기서 요청 생성을 두 경우 모두 0.1초 수준으로 한 것은 본 실험이 확장성에 관한 문제만을 제시한다는 것을 보이기 위해서이다. 다시말해 요청 빈도는 같은 조건에서 단지 캐쉬 수의 변화에 따른 결과를 보이기 위해서이다.

결과를 보면 요청수가 적을 때(request 개수 : 1,000)는 반응시간에 차이가 거의 나타나지 않았다. 오히려 중첩 캐쉬가 더 느린 평균 반응 시간을 나타내 주고 있는데 이는 요청수가 적어 아직 적응성 반응(계층 캐싱의 임계값:5, 중첩 캐쉬의 TTL값 : 1초)이 본격적으로 일어나지 않았기 때문이다. 요청 수가 증가(request 개수 : 10,000)하면서 적응성 페이지 복사가 이루어지자 캐쉬 개수의 증가, 즉 확장성이 커지자 중첩 그룹 캐쉬 방식이 보다 빠른 평균 반응시간을 나타내 주고 있다. 이는 계층 캐쉬의 경우 캐쉬의 개수가 늘어남에 따라 하부 수준에서 ICP를 통한 멀티캐스트 요청시 overhead가 지수적으로 증가하는데서 그 원인을 찾을 수 있다. 즉 확장성에 한계를 나타낸 것이라 볼 수 있다. 반면 중첩 캐쉬의 경우 이러한 지연이 그리 늘어나지 않는 것을 볼 수 있는데 이는 캐쉬 개수가 늘어나면 적응성 반응을 통해 중첩 캐쉬가 계층 캐쉬보다 확장성 면에서 더 유리하다는 것을 보여 주고 있다.

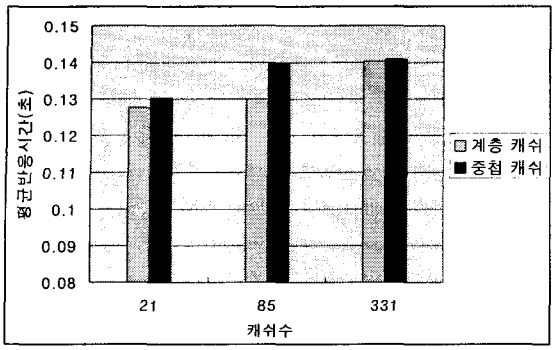


그림 18. 요청수 1,000개일 때 반응시간

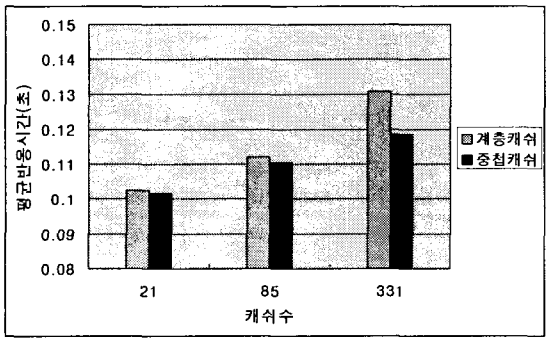


그림 19. 요청수 10,000개일 때 반응시간

V. 결론

우리는 본론에서 동적 캐쉬 구조의 필요성과 그를 위한 중첩 캐쉬 그룹을 제시하였다. 이를 위해 두 가지 특징적 구조를 제시하였는데 역할 캐쉬와 group ID가 그것이다. 역할 캐쉬는 중첩 구조에서 멀티캐스팅 방식의 복사를 통해 야기되는 저장 장소의 낭비를 줄이는 기능과 그룹간의 통신 연결 역할을 담당하며, group ID는 그룹 유지와 요청 진행 방향 결정시에 사용되는 요청 진행 테이블의 구성 요소로 작용하게 된다. 한편 효과적인 요청 진행 방향을 제시하기 위해 요청 진행 테이블을 유지하였으며 이 테이블에 TTL 기법을 사용함으로써 동적인 요청 진행이 가능하게 해 줄 수 있었다. 결국 중첩된 구성 기법은 동적인 캐쉬 구성을 가능하게 해 줄 수 있는 구조이며 이를 통해 캐쉬의 공유에서 가장 큰 문제로 지적되는 확장성 문제에 대한 해결책을 제시할 수 있다.

참 고 문 헌

[1] P. B. Danzig, R. S. Hall, M. F. Schwartz "A Case for Caching File Objects Inside Internetworks," in Proceeding of the ACM SIGCOMM'93, Sep. 1993.

[2] David Karger,, Eric Lehman, etc. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in Proceedings of the ACM STOC'97, 1997.

[3] L. Fan, P. Cao, J. Almeida and A. Z. Broder. "Summary Cache: A scalable wide-area cache sharing protocol," in Proceedings of the ACM SIGCOMM'98, Oct. 1998.

[4] Radhika Malpani, Jacob Lorch and David Berger. "Making World Wide Web Caching Server Cooperate," In Proceedings of World Wide Web Conference, 1996.

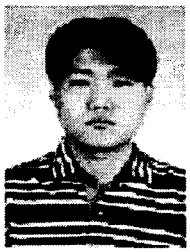
[5] A.Chankhunthod, P.B. Danzig, C.Neerdals, M. F. Schwartz and K. J. Worrell. "A Hierarchical Internet Object Cache," in Proceedings of the USENIX Technical Conference, SanDiego, CA, Jan. 1996.

[6] Lixia Zhang, Sally Floyd, and Van Jacobson. "Adaptive web caching," In the 2nd Web Caching Workshop, oulder, Colorado, Jun. 1997.

[7] Haobo yu, Lee Breslau, "A Scalable Web Cache Consistency Architecture," in Proceedings of the ACM SIGCOMM'99, 1999.

[8] M. F. Arlitt, Carey L. Williamson, "Web Server Workload Characterization: The Search for Invariants," in Proceedings of the ACM SIGMETRICS, May 1996.

현 진 일(Jin-il Hyun) 정회원



1999년 2월 : 동국대학교 컴퓨터공학과 (공학사)
2001년 2월 : 동국대학교 컴퓨터공학과 (공학석사)
1999년 3월 ~ 현재 : 명세정보시스템 기술연구소 연구원

<관심분야> : 컴퓨터구조, 인터넷, 네트워크

민 준 식(Jun-Sik Min) 정회원

1994년 2월 : 동국대학교 컴퓨터공학과 (공학석사)
2001년 8월 : 동국대학교 컴퓨터공학과 (공학박사)
1994년 ~ 1995년 : 쌍용정보통신 연구원
1996년 ~ 2001년 : 한국전산원 국가망이용관리팀장
2002년 3월 ~ 현재 : 경동대학교 정보통신공학부 전임강사

<관심분야> : 병렬 및 분산처리, 컴퓨터구조, 망관리