캐시의 역할 구분을 이용한 확장성이 있는 캐시 그룹 구성 정책

현진일[U], 장태무
동국대학교 컴퓨터 · 멀티미디어 공학과
{littleh, jtm}@dongguk.edu

A Scalable Cache Group Configuration Policy using Role-Partitioned Cache

Jinil Hyun[U], Teamu Chang
Dept. of Computer · Multimedia Engineering, Dongguk University

요 약

동적 캐시 그룹 구성 정책으로 중첩된 그룹 캐시 구조가 제시되면서 중첩 캐시의 운영 방안에 대한 연구가 필요하게 되었다. 그 기반 연구로 계층 캐시의 효율적 캐시 분배 정책을 그대로 살리되 중첩 그룹 캐시를 이용하는 방안으로 그룹 내 멀티캐스팅 페이지 분배 정책이 제시되고 있다. 그러나 그룹내의 모든 캐시에 해당 페이지를 분배하는 방안은 불필요한 저장 공간 낭비를 가져 올 뿐 아니라 그룹 내 통신량 증가라는 문제점을 내포하고 있다. 따라서 본 논문에서는 그룹 내 캐시의 역할을 두 개의 기능으로 나누어 보다 효율적 페이지 분배와 그룹 유지 운영 방안을 제시하려 한다. 또한 중첩 그룹 캐시에서 제기되는 요청 진행 방향 문제에 대한 해결 방안으로 그룹 아이디를 활용한 진행 방향 테이블 운영 방안을 제시하려 한다.

1. 서론

인터넷 환경이 급속히 확장하는 가운데 서버의 부하 및 네트워크내의 통신량의 감소, 반응 지연 시간의 감소 등의 목적으로 캐시의 중요성이 대두되고 있다. 실제로 캐시를 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 캐시 운영 정책으로 현재 harvest, squid 방식과 이를 응용한 방법들이 널리 사용되고 있다. 그러나 위에서 보인 캐시 구조들은 정적인 형태로 구성되어 있고 이러한 구조로는 급속히 확장하는 캐시를 감당하는 데에 한계가 있다. 다시 말해 정적구조에서는 고정적 경로를 따르는 데이터 흐름으로 인해 순간적인 요청 몰림 현상(hot spot)[2]에 대한 대처가 어렵고 또한 구조 유지를 위한 정보 교환은 확장시 전체적인 통신량을 증가시켜 확장성을 떨어뜨리게 된다. 이에 동적인 캐시 구조를 요구하게 된다. 본 논문에서는 이러한 동적 캐시구조로 나아가기 위한 캐시 정책을 제시하려 하는데 그 방법이 역할 캐시를 이용한 중첩 그룹 캐시의 구조이다. 이 구조상에서 캐시는 두 가지 자격을 갖는데 그룹 내 캐시(ingroup cache)와 중첩 캐시(overlab cache)가 그것이다. 그룹 내 캐시는 기존의 캐시 역할만을 담당하고 여기에 덧붙여 중첩 캐시는 그룹과 그룹의 연결 역할을 수행하게 된다.

2. 기존 연구

초기 인터넷 캐시는 서버와 클라이언트 사이의 proxy 캐시로써 독점적 역할을 담당하였다. 인터넷 환경의 확장은 이들 캐시 사이의 공유를 요구되게 되었고 Harvest project는 ICP를 통해 주변 캐시들의 내용을 공유하는 방법을 제시하여 처음으로 "Web cache sharing" 개념을 도입하였다. 그러나 ICP를 이용한 캐시 사이의 공유는 캐시 수가 증가할수록 통신량의 증가와 반응 지연시간(latency)의 증가를 가져와 확장성에서의 한계를 드러내게 된다. 이를 극복하는 방법으로 계층 캐시 구조[3]와 지역적으로 그룹을 나누는 방법 등이 제시되게 된다. 이중 널리 사용되는 계층 캐시구조는 클라이언트가 접속한 캐시에서 미스가 이루어 질 경우 같은 수준(level)의 캐시들을 ICP (Internet Cache Protocol)를 통해 검색하고 여기서도 미스가 이루어 질 경우 상위 수준(parent cache)에 다시 요청하는 방향으로 요청 진행이 이루어지는 방식이다. 그러나 이러한 구조는 상위 수준의 캐시로 갈수록 병목 현상을 유발하여 순간적 요청에 대해 반응성이 떨어지는 결과를 가져온다. 이는 요청이 빈번한 페이지를 계층의 하부 즉 클라이언트에 가까이 두어 요청 흐름이 상위 수준까지 올라가는 것을 막음으로써 해결이 가능하다. 이러한 방식을 적응성 계층구조라 하겠다. 적응성 캐시 구조의 구현은 일정 기간 동안에 요청량이 어느 기준치를 넘으면 하위 수준의 캐시에 페이지 복사가 이루어지고 다음 흐름의

요청은 복사가 이루어진 캐시에서 담당하게 된다. 여기서도 요청량이 기준치를 넘으면 하위 수준의 캐시에 페이지 복사가 이루어지고 이 과정을 반복하여 클라이언트에 가까운 위치에 해당 페이지를 위치시키게 되는 것이다. 이 논문에서는 이러한 적응성 캐시의 장점을 그대로 차용하였다.

3. 본론

여기서 주장하고자 하는 내용은 기존의 계층(hierarchy) 구성 방식 캐시의 장점은 그대로 살리되 동적 구성방식의 구현이 가능한 중첩된 캐시 그룹이다.

3.1. 캐시 운영 정책

요청(request) 측면에서 바라본 흐름을 살펴보자. 클라이언트에 의해 들어온 요청은 일단 자신의 지역 캐시를 탐색하게 된다. 여기서 미스가 일어나면 다음으로 그 캐시가 속한 그룹의 다른 캐시들을 찾게 되는데 이때 두 가지 방법을 생각할 수 있다. 멀티캐스팅 방식과 해싱 테이블을 이용한 탐색 방식이다. 해싱 테이블을 이용하는 방식은 각 캐시가 자신이 속한 그룹의 다른 캐시들이 저장하고 있는 페이지 정보를 유지함으로서 이를 통해 바로 요청한 페이지가 있는 캐시를 찾아가는 방식이다. 이러한 방식을 채택할 경우 중첩 캐시는 여러 그룹의 캐시 목록 테이블을 유지하여야 하는 문제가 발생하게 된다. 따라서 여기서는 멀티캐스팅 방식을 사용하려 한다. 이 경우 그룹 내 통신이 많아지는 단점이 있지만 그룹 사이즈가 작을 경우 그룹 내 통신량이 그리 크지 않게 되며 각 캐시가 지역 내 다른 캐시들의 정보를 유지하여야 하는데 드는 비용도 없게 된다. 다음 단계로 이렇게 하여 해당 그룹에서 적중(hit)이 일어나지 않을 경우 요청은 기본적으로 해당 그룹의 중첩 캐시에 보내게 되고 여기서 다른 그룹으로의 요청 전이가 일어나게 된다. 이런 과정을 거듭하여 캐시 미스가 일어나면 최종적으로 서버에서 반응(response)이 이루어지게 된다.

다음으로 반응 측면에서의 흐름을 살펴보자. 서버까지 올라온 요청은 서버에서 적중이 일어나고 반응이 이루어지게 된다. 한편 적응성 캐시 구조를 형성하기 위해서는 반응이 이루어진 페이지를 단지 클라이언트에 전달하는 것에서 그칠 것이 아니라 하부 캐시에도 페이지 저장이 이루어져야 한다. 기존에 발표된 논문[4]에서는 반응이 일어난 그룹의 모든 캐시에 반응 페이지가 멀티캐스팅 방식으로 전달되게 되고 반응그룹 내 모든 캐시들이 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 캐시에 너무 많은 저장 장소를 요구하게 된다. 따라서 본 논문에서는 적중이 일어난 그룹 내의 모든 캐시에 반응 페이지가 멀티캐스팅 되는 방식이 아닌 해당 그룹의 중첩 캐시에만 멀티 캐스팅 되는 방식을 택하려 한다. 이렇게 하면 다음 흐름의 요청에 대하여 경로를 타고 올라온 경우에는 중첩 캐시에서 적응성(adaptive)있게 반응을 하게 되고 해당 그룹이 클라이언트로부터 접속되는 캐시가 존재하는 그룹이라면 해당 그룹 내 캐시에서 바로 적중이 일어나지 않고 서버에서 반응하게 된다. 그러나 이것은 한번의 미스(1 hop의 overhead)이고 반응시의 통신 량을 줄임으로써 보다

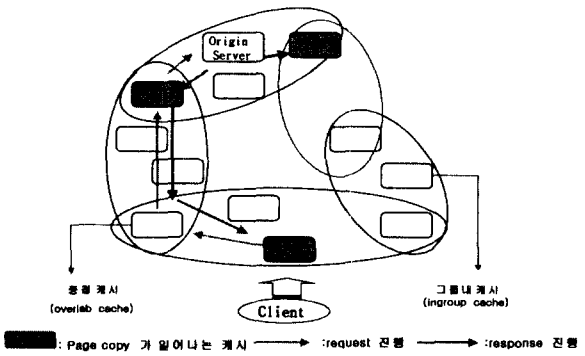

그림1. 중첩 그룹 캐시 구조

효율적인 공간 유지와 통신 복잡성을 줄일 수 있다(요청시의 패킷은 요구 시 필요한 정보(원하는 페이지 정보)만을 가지고 있지만 반응 시에는 해당 페이지 자체를 이동시켜야 하므로 반응시의 데이터 이동량이 요청시보다 훨씬 많다.). 여기서 제기될 수 있는 문제가 그룹 내 캐시들이 캐시 기능 상실의 문제이다. 반응된 페이지가 중첩 캐시에만 저장된다면 그룹 내 다른 캐시들은 캐시 기능을 발휘 할 수 없게 된다. 그래서 적중이 일어난 그룹의 중첩 캐시뿐 아니라 클라이언트가 접속한 캐시에도 해당 페이지를 내용을 저장하려 한다. 클라이언트가 접속한 캐시는 중첩 캐시 뿐 아니라 그룹내의 모든 캐시일 수 있는 것이기에 이렇게 함으로써 그룹 내 캐시도 캐시 기능을 수행할 수 있다. 이를 구현 할 수 있는 방법으로 각 요청 흐름마다 계수기(counter)를 달아 count[hit](hit는 적중 일어난 그룹까지의 총 그룹의 경로 수)인 곳에서는 중첩 캐시에 멀티캐스트를 통한 캐싱이 이루어지고 count[0](client의 요구를 받은 캐시)인 곳에서는 일반적인 캐싱이 이루어지게 하는 것이다.[그림 1]

한편 적응성 캐시를 구성하기 위해서는 각 캐시에 캐싱 되는 페이지들을 어떻게 유지하여야 하는 문제도 고려하여야 하는데 여기서는 가장 일반적이고 효과적인 TTL(Time to Live) 방식을 사용하려 한다.

3.2 중첩 캐시의 구현

중첩 캐시가 가져야할 구조를 생각해 보자. 중첩 캐시가 그룹 내 캐시와 다른 가장 큰 특징은 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. 먼저 각 그룹을 구별하기 위해 group ID라는 개념을 도입하려 한다. group ID는 그룹 내 캐시에서는 자신이 속한 그룹을 확인하는 요소로 사용되고 요청 메시지를 멀티캐스트 할 때는 메시지가 도달하는 범위 한정 역할을 한다. group ID를 사용할 때 중요하게 고려되어야 하는 사항을 살펴보자. 그룹이 생성되거나 소멸 또는 합쳐질 경우 기존 그룹들의 ID는 변화가 필요하다. 이 경우 일반적으로 생각 할 수 있는 것이 중앙 집중적인 방식으로 group ID를 관리하는 서버를 통해 그룹 생성 시는 새로운 ID를 부여하고 소멸 시는 해당 ID를 환수하며 결

합 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 채택 그룹의 변화에 적절한 ID 운영정책을 성립 하려한다. 이 경우 그룹간에 정보 유지를 위해 발생하는 통신량 증가는 페이지 요청 시 해당 정보를 piggyback 형식으로 처리하여 최소화하려 한다. 즉 그룹 생성 시는 주변 정보를 통해 사용되지 않는 ID 하나를 생성하고 소멸 시는 자신의 ID 소멸을 주변 그룹에게 알려주면 될 것이다. 한편 결합 시는 두 ID를 모두 소멸하고 새로운 ID를 공통으로 부여받는 방법을 고려 할 수 있으나 이 경우 두 그룹의 모든 캐시의 group ID를 변경하여야 한다. 이 경우 두 그룹의 크기(ex: 캐시 수)를 비교하여 큰 그룹의 ID를 승계한다면 이에 드는 비용을 최소 반 이상을 줄일 수 있다.

3.3 request forwarding

여러 그룹이 중첩되어 존재하다 보면 원하는 페이지를 찾아 들어가는 경로가 여럿이 존재할 수 있다. 이러할 경우 전체 그룹 내에 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 경로 방향 진행 테이블을 중첩 캐시에 구현함으로써 해결하려 한다. 반응(response)되어 돌아오는 메시지는 자신이 거쳐간 경로를 그대로 따라오게 된다. 이때 각 중첩 캐시가 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 최단 경로의 요청 진행이 이루어질 수 있다. 뿐만 하니라 전체 경로를 저장하는 것 보다 저장 장소의 낭비를 최소화 할 수 있으며 TTL[5] 개념을 도입, 일정 간격으로 페이지 진행 테이블을 갱신한다면 동적인 요청 흐름 제어가 가능하다고 본다.

4. 구현 및 성능 평가

구현은 펜티엄 PC의 Windows 2000 운영체제상에서 이루어졌으며 시뮬레이터로써는 ns-2(Network Simulator-2)가 사용되었다.

본 논문의 목적이 중첩 캐시 구조를 이용한 동적인 확장성 캐시 구조를 이루는 것인 만큼 이를 보이기 위해 전체 캐시수를 등차적으로 (100~1,000)증가시켜 가면서 측정하였다.

ns-2 시뮬레이터를 통해서 임의적인 trace를 생성하였고 각 형성 페이지의 크기는 일반적인 페이지 크기인 100~10,000 바이트[6] 사이로 제한하였다. 이를 기반으로 기존의 ICP를 이용한 정적인 계층캐시(4-nary)와 동적 중첩 그룹 캐시(그룹의 크기를 20으로 고정)의 평균 반응시간(average response time)을 비교하였는데 캐시 수가 적을 때는 반응시간에 차이가 거의 나타나지 않았으나 캐시의 수가 증가함에 따라 중첩 그룹 캐시 방식이 보다 빠른 평균 반응시간을 나타내 주고 있다.

5. 결론

우리는 본론에서 동적 캐시 구조의 필요성과 그를 위한 중첩 캐시 그룹을 제시하였다 .. 이를 위해 두 가지 특징적 구조를 제시하였는데 역할 캐시와 group ID 이다. 역할 캐시는 중첩 구조에서 야기되는 저장 장소 낭비를 줄이는 기능을 group ID는 그룹 유지와 요청 방향을 명확히 하는 기능을 담당하게 된다.

6. 향후 과제

본 논문의 궁극적인 목적은 동적 캐시 구조를 형성하는 것이다. 그러기 위해서는 그룹 내 캐시에서 각 개별 캐시가 참여(join)하거나 제거(delete) 되는 과정이 필요하다. 참여, 제거는 그룹의 크기와 상황에 따라 변화시키는 것으로 이를 위해서는 그룹상의 캐시 개수, 그룹 내 통신량 등의 그룹 정보를 유지하여야 한다. 이러한 그룹 정보는 각 캐시들이 모두 가지고 공유하여야 하며 이러한 정보 공유방식에 대한 연구가 더 필요하다고 본다. 또한 일관성 유지의 문제도 고려하여야 하겠다. 서버가 위치한 그룹에서의 캐시들에 대해서는 일관성 유지가 쉽겠지만 해당 페이지를 가지고 있는 다른 그룹의 캐시들에 대해서는 일관성 유지 문제를 심각하게 생각하지 않을 수 없다. 왜냐하면 해당 페이지가 어느 그룹에 흩어져 분포되었는지에 대한 정확한 정보를 파악하는 게 힘들기 때문이다. 과다 중첩의 문제 또한 중첩 캐시를 구성하는데 있어 풀어야 할 과제이다. 과다 중첩의 경우 중첩 캐시는 자신이 속한 많은 그룹의 정보와 집중된 캐싱 페이지를 유지하여야 한다. 이러한 문제에 대해서 bloom filter[7]나 차별화된 TTL 방식을 통해 해결책을 모색하여야 하겠다.

참고 문헌

[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 September, 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]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, San Diego, CA, January 1996

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

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

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

[7]L. Fan, p. Cao, J. Almeida and A. Z. Broder. "Summary Cache: A scalable wide-area cache sharing protocol" in Proceed-ings of the ACM SIGCOMM' 98, October, 1998.