액티브 네트워크 환경에서 실행 코드 교체를 위한 효율적인 캐싱 기법

조정희, 김동혁, 장창복, 조성훈, 최의인
한남대학교 컴퓨터공학과

Efficient Caching Technique for Replacement of Execution Code on Active Network Environment

Jung-Hee Cho, Dong-Hyuk Kim, Chang-Bok Jang, Sung-Hoon Cho, Eui-In Choi
Dept. of Computer Engineering, Hannam University

요 약

인터넷의 급속한 발전과 컴퓨터 성능의 발달로 많은 사용자들은 네트워크를 통해 정보를 획득하고 이용하고 있다. 이에 따라 사용자의 요구도 빠르게 증가하고 있으며, 이러한 사용자 요구를 해결하기 위해 액티브 네트워크와 같은 기술들이 활발하게 연구되고 있다. 액티브 네트워크란 라우터나 스위치가 프로그램 실행 능력을 가지고 있어서 프로그램을 포함하고 있거나 중간 노드의 프로그램을 실행하도록 하는 패킷을 다양하고 유동적으로 처리할 수 있는 환경을 말한다. 이러한 액티브 네트워크의 중간 노드(Active Node)는 단순한 패킷 전달(forwarding) 기능 이외에 사용자의 실행 코드를 저장하고, 처리할 수 있는 기능을 가지고 있다. 따라서 액티브 노드에서 패킷을 실행하기 위해서는 각 패킷을 처리하는데 필요한 실행 코드가 요구되고, 이러한 실행 코드는 이전의 액티브 노드나 코드 서버에 요청함으로써 얻을 수 있다. 하지만 이러한 실행 코드를 이전 액티브 노드나 코드 서버에서 가져오게 되면 실행코드가 전달될 때까지의 시간지연이 발생하므로 사용되었던 실행 코드를 액티브 노드의 캐시에 저장하여 코드의 실행 속도를 증가 시킬 필요가 있다. 따라서 본 논문에서는 액티브 노드 상에 실행 코드를 효율적으로 캐시 함으로써 실행 코드 요청의 횟수를 줄이고 패킷 처리 속도를 향상시킬 수 있는 캐싱 기법을 제안하였다.

1. 서론

인터넷이 급격하게 확산되고 인터넷을 이용하는 사용자가 늘어남에 따라서 네트워크에 대한 요구는 점점 복잡해지고 있다. 하지만 현재의 네트워크 시스템은 새로운 기술이나 표준을 네트워크 망에 적용하기까지 많은 시일과 비용이 소요된다. 또한 사용자 요구 조건의 변화 속도가 빠르게 변화하는 것에 비해 이를 지원하는 네트워크 시스템은 상대적으로 변화 속도가 느리기 때문에 사용자의 망에 대한 요구사항을 시기적절하게 반영하는 것이 불가능하다. 따라서 이러한 문제점을 해결하고자 연구되고 있는 분야가 액티브 네트워크(Active Network)이다. 액티브 네트워크란 라우터나 스위치가 프로그램 실행 능력을 가지고 있어서 프로그램을 포함하고 있거나 중간 노드의 프로그램을 실행하도록 하는 패킷을 다양하고 유동적으로 처리할 수 있는 환경이다[1]. 이러한 액티브 네트워크 연구로는 Switchware, ANTS, CANE, FAIN과 같은 것들이 있다[6, 7]. 또한 기존 네트워크가 중간 노드에서는 단순히 패킷의 경로를 설정하고 전달하는 기능을 담당하고, 에러처리 및 흐름제어와 같은 패킷의 복잡한 처리는 종단의 단말 장치에서만 처리하던 것과는 달리 액티브 네트워크의 중간 노드에서는 다양한 작업을 처리할 수 있도록 함으로써 기존의 망에서 제공하지 못했던 유연성과 다양한 장점을 제공할 수 있다[2].

이러한 액티브 네트워크 상에서의 중간 노드(또는 액티브 노드)는 노드에 도착한 패킷을 분류하고 패킷(또는 액티브 패킷)내에 포함된 코드를 해석하고 실행한 후에 그 결과를 다음 노드에 전달하는 역할을 수행한다. 따라서 액티브 노드에서 패킷을 처리하기 위해서는 처리할 데이터와 실행 코드가 필요하며 실행 코드는 노드에 적재되어 있어야 한다. 만약 노드에 실행코드가 존재하지 않으면, 다른 노드나 노드 서버로부터 코드를 요청하여 전달 받아야 한다. 따라

본 연구는 한국과학재단 목적기초연구(R01-2002-000-00127-0) 지원으로 수행되었음

서 실행에 필요한 코드가 실행될 노드에 전달되기까지 시간상의 지연이 발생할 수 있다. 그러므로 실행 코드를 노드내의 캐시에 저장하여 실행 지연시간을 줄일 수 있는 캐시 기법에 관한 연구가 필요하다.

따라서 본 논문에서는 액티브 노드에서 패킷을 처리하기 위해 필요한 실행 코드를 캐시에 저장한 후 참조 횟수와 시간 제약(time limit)을 통해 좀더 효율적으로 캐시가 가능한 기법을 제안한다.

2. 관련 연구

2.1 액티브 네트워크

가. 액티브 네트워크 구조

액티브 네트워크의 중간 노드인 스위치나 라우터는 단순한 패킷 전달(forwarding) 기능 이외에 사용자의 실행 코드를 저장/처리/전달 할 수 있다. 사용자는 프로그램 코드를 전송하여 실행하거나 네트워크 노드에 미리 설치된 프로그램을 호출함으로써 자신이 원하는 네트워크 기능을 이용할 수 있게 되었다[1, 2]. 그림 1은 액티브 네트워크의 구조를 보여주는 그림이다.

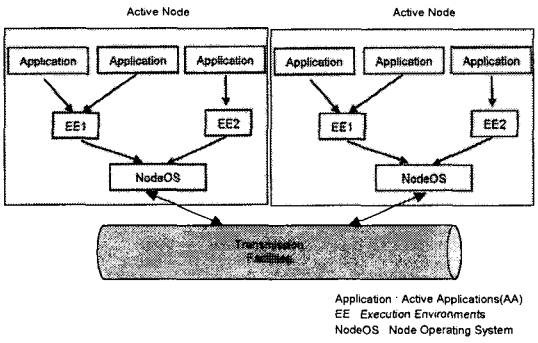

그림 1. 액티브 네트워크 구조

액티브 네트워크는 일반적으로 기존 네트워크의 패킷에 해당하는 캡슐과 이를 중간에서 처리할 수 있는 액티브 노드, 그리고 캡슐을 생성할 수 있는 액티브 서버로 이루어져 있다. 액티브 노드는 액티브 네트워크를 구성하는 가장 기본적인 구성요소로서 노드에 도착한 패킷에 대해 액티브 패킷을 분류하고 액티브 패킷 내에 포함된 코드를 해석하여 실행한 후에 실행 결과를 다음 노드로 전달하는 기능을 담당하며, 노드 운영 체제, 실행환경, 액티브 응용(Active Application : AA)으로 구분된다. 노드 운영체제는 액티브 패킷을 전달 받아 이를 적절한 실행환경으로 분배하고, 실행환경에서 해당 액티브 패킷을 처리하기 위해 요구되는 자원을 관리(할당, 제어, 반환)하며, 실행환경으로부터의 결과를 다른 노드로 전달하는 기능을 수행한다. 실행 환경은 일종의 가상 기계(virtual machine)로 노드 운영체제로부터 전달 받은 액티브 패킷 내에 포함된 코드를 해석하고 실행한 후에 그 결과를 노드 자신의 액티브 응용이나 노드 운영체제로 전달하는 기능을 수행한다. 이때 실행에 필요한 실행코드는 노드에 적재 되어 있어야 하며 만약 필요한 실행코드가 노드에 존재하지 않을 경우 다른 노드로부터 전달받아야 한다. 이러한 경우에 요구되는 기술이 코드 요청 기술로써, 현재 이전 액티브 노드로부터 전달받는 방법과 코드 서버로부터 전달 받는 방법이 연구되고 있다[4].

나. 코드 요청 기술

그림 2는 ANTS의 코드 분배 프로토콜을 나타낸 것이며, 액티브 노드에서 패킷을 처리하기 위해 실행 코드 요청이 어떻게 이루어지는지를 설명하고 있다[3]. 액티브 노드가 액티브 패킷을 처리하는 동작은 다음과 같다.

- 액티브 패킷이 액티브 노드에 도착하면, 노드는 액티브 패킷을 처리하기 위한 코드가 캐시에 존재하는지 검사한다.
- 코드가 캐시에 존재하지 않으면 지나온 가장 이전의 노드에게 코드를 요청하고, 캡슐의 실행은 한정된 시간동안 연기되고 캡슐은 대기 상태가 된다.
- 코드 전송 요구에 대한 응답을 받으면, 코드를 캐시에 넣고, 대기 상태의 액티브 패킷을 처리한다. 만약 일정 시간이 지나도 요청에 대한 응답이 도착하지 않거나, 응답이 필요한 것이 아니라면 캡슐은 버려지게 된다.

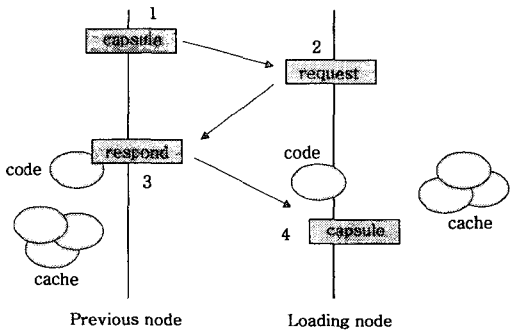

그림 2. ANTS의 코드 분배 프로토콜

따라서 좀더 효율적인 코드 실행을 위해서는 실행된 코드가 액티브 노드내에 존재해야 하기 때문에, 실행된 코드를 캐시에 저장하기 위한 효율적인 캐싱 기법이 필요하다.

2.2 기존의 캐싱 기법

기존의 캐시된 데이터의 교체 알고리즘으로는 FIFO(First-In-First-Out), LFU(Least Frequently Used), LRU (Least Recently Used) 기법 등이 있다[5]. FIFO 기법은 교체될 대상을 선택함에 있어 가장 먼저 들어온 것을 선택하여 교체하는 기법이다. 이 기법은 이해하기 쉽고 프로그램하기 쉽지만, 실행 코드를 캐시에 저장할 때 적재 공간이

부족하여 코드 중에 하나를 삭제하려 할 경우, 사용 빈도수가 높음에도 불구하고 가장 먼저 적재된 코드라면 삭제되어야하는 문제점이 존재한다. 즉, 일반적으로 사용 빈도수가 높은 코드는 다시 요구될 가능성이 많기 때문에 바로 이전에 삭제된 코드를 다시 요구할 수 있어 비효율적이다. LFU 기법은 데이터가 얼마나 많이 참조되었는지에 따라 교체할 데이터를 찾는 방법이다. 이 기법은 교체가 필요한 경우에 참조 횟수가 가장 적은 것을 교체하며, 자주 사용되는 코드를 계속 유지함으로써 좋은 성능을 나타낼 수 있지만 단기간에 많이 사용되고 오랜 시간동안 사용되지 않을 경우에도 캐시에 코드가 유지되는 단점이 있다. LRU 기법은 가장 오랫동안 참조되지 않은 것을 교체하는 기법으로, 최근에 참조되었는가를 기준으로 교체하기 때문에 참조 횟수를 반영하지 못하는 단점을 가지고 있다.

3. 제안한 캐싱 기법

액티브 네트워크에서는 동적으로 패킷을 처리하기 위한 실행 코드가 다양하게 존재하며, 이러한 실행 코드를 효율적으로 캐시에 유지함으로써 코드 요청 횟수를 줄일 수 있다. 코드 요청 횟수의 감소는 패킷 처리 대기 시간의 감소와 패킷 처리 속도 증가를 가져올 수 있다.

따라서 본 논문에서는 이러한 실행코드를 좀더 효율적으로 캐시하기 위해 실행 코드의 실행 횟수와 시간 제약(time limit)에 대한 필드를 코드 정보 테이블(Code Information Table)에 저장하고 관리하여 효율적으로 실행코드를 캐시에서 교체할 수 있도록 하였다. 이러한 코드 정보 테이블은 그림 3과 같이 각 코드에 대한 식별자 필드(Code ID)와 코드 실행 횟수에 대한 정보 필드(Execution No), 시간 제약 필드(Time Limit No)로 구성되어져 있다.

Code ID	Execution No	Time Limit No
A	5	1
B	1	1
C	7	0

그림 3. 코드 정보 테이블의 구성

코드 실행 횟수 필드는 코드가 최초 실행될 때 '1' 값을 가지며, 재실행 될 때마다 값이 '1'씩 증가된다. 시간 제약 필드는 시간 제약 값에 대한 정보를 주기적으로 갱신하여 유지한다. 코드가 최초로 실행된 후에 코드는 시간 제약에 대한 일정한 값(예:10)을 가지게 된다. 캐시 관리자는 코드가 실행된 후 일정 시간 마다(예:1시간) 시간 제약 값을 '1'씩 감소시킨다. 시간 제약 값은 코드가 재실행 될 때 다시 초기 값(10)을 가진다. 캐시에 저장되어져 있는 실행 코드를 삭제하고 새로운 실행 코드를 적재하기 위해서는 실행코드를 교체하기 위한 알고리즘이 필요하다. 본 논문에서는 실행코드 교체 알고리즘으로 두 가지 방법을 제안하였다.

첫 번째 방법은, 일정한 시간에 따라 감소되는 시간 제약 값이 0이 되는 실행 코드를 바로 삭제하지 않고 캐시 내에 유지시켜, 교체가 필요한 경우 시간 제약 값이 0인 실행 코드들 중에 참조 횟수가 가장 작은 것을 먼저 삭제한다. 이러한 방법은 참조 횟수가 많은 코드임에도 불구하고 시간 제약 값이 0이 되어 바로 삭제되는 문제점을 방지 할 수 있다. 두 번째 방법은 실행코드 교체 요구 시 시간 제약 값이 0 인 실행 코드가 하나인 경우, 실행 횟수를 이용하지 않고 시간 제약 값이 0인 코드를 삭제하는 기법이다. 이 기법은 단기간에 참조되고 오랫동안 참조되지 않는 실행코드들을 삭제하여 캐시 공간의 효율성을 증가시킨다. 그림 4는 제안한 기법의 동작순서를 순서도로 나타낸 것이다. 제안한 알고리즘의 동작 과정을 살펴보면 다음과 같다. 이때의 동작 과정은 패킷 실행에 필요한 코드 요청에 대한 응답을 받은 상황으로 가정한다.

① 캐시 관리자는 코드를 캐시에 넣기 위해 캐시에 공간이 있는지를 검사한다.

② 캐시에 공간이 있으면 코드를 캐시에 적재한다. 만약 필요한 공간이 부족하면, 캐시 관리자는 시간 제약 필드를 검사한다.

③ 검사된 코드 중에서 시간 제약 값이 0인 경우가 여러 개이거나 0인 경우가 없을 때에는, 실행 횟수가 가장 적은 코드를 삭제한다.

④ 검사된 코드 중에서 시간 제약 값이 0인 경우가 하나일 때는, 해당 코드를 삭제한다.

⑤ 캐시 관리자는 코드 적재를 위해 필요한 공간이 생길 때 까지 이 과정을 반복한다.

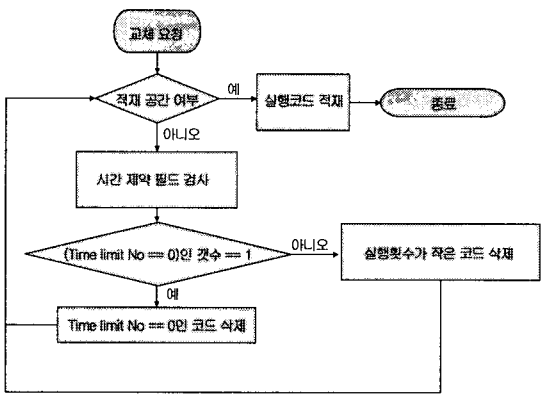

그림 4. 제안한 기법의 동작 과정

예를 들어 그림 3의 코드 정보 테이블처럼 실행 코드 A, B, C가 캐시 내에 저장되어 있고, 실행 코드 D를 캐시에 적재하려고 할 경우, 코드 적재 공간이 부족하기 때문에 실행코드를 삭제해야 한다. 따라서 A, B, C 코드 중에서 시간 제약 값이 0인 경우가 하나이기 때문에 본 논문에서 제안한 두 번째 기법에 의해 C 코드를 삭제하고 D 코드를 적재하게 된다. 만일 1시간이 지난 후(시간 제약 값이 1 감소) 새로운 코드 E가 캐시에 적재될 경우, 시간 제약 값이 0인 코

드가 A, B 이기 때문에(1시간동안 실행이 되지 않았다고 가정), 본 논문에서 제안한 첫 번째 기법에 의해 삭제될 코드는 실행 횟수가 작은 B 코드가 된다. 따라서 최종 코드 정보 테이블은 다음 그림 5와 같이 된다.

Code ID	Execution No	Time Limit No
A	5	0
E	1	10
D	1	9

그림 5. 실행 코드 삭제 이후의 코드 정보 테이블

제안한 알고리즘에서는 자주 실행되는 코드가 일정 시간이 지난 후 바로 삭제되는 것을 방지하기 위해 시간 제약 값에 따라 적용되는 캐시 교체 기법을 다르게 하여, 자주 실행되는 코드를 캐시에 유지시키고, 장시간 실행되지 않는 코드를 삭제할 수 있도록 함으로써 코드 요청 횟수를 줄이고 캐시 공간을 좀더 효율적으로 사용할 수 있도록 하였다.

4. 결론

본 논문에서는 액티브 노드 상에서 동적으로 패킷을 처리하기 위한 실행 코드 요청 횟수를 줄이기 위해, 액티브 노드의 캐시에 실행코드를 저장하여 사용할 수 있도록 하였고, 캐시내의 실행 코드 교체 시 효율적인 교체 알고리즘을 제안함으로써 코드 요청 횟수와 캐시 공간 낭비를 줄일 수 있다. 따라서 제안된 캐싱 기법을 사용함으로써 액티브 노드 상에서 패킷 처리 대기 시간을 감소시키고 패킷 처리의 속도 향상을 기대할 수 있다.

향후 연구 과제로 다른 기법과의 비교를 통한 성능의 검증이 필요하며 액티브 노드 구조와 패킷 실행 환경에 대한 연구를 병행하여 더 나은 패킷 처리 속도 향상을 위한 캐시 기법에 대한 연구가 필요하다.

[참고문헌]

[1] D. L. Tennenhouse, J. M. Smith, W. D Sncoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research," IEEE Communications Magazine, Vol. 35, No. 1, pp. 80-86, 1997.

[2] D. L. Tennenhouse, D. J. Wetherall, "Towards an Active Network Architecture," Multimedia Computing and Networking, January 1996.

[3] David J. Wetherall, John V. Guttang and David L. Tennenhouse, "ANTS : A Toolkit for Building and Dynamically Deploying Network Protocols," IEEE OPENARCH, 1998.

[4] 안상현, 김경춘, 손선경, 손승원, "능동 응용의 특성을 고려한 능동 노드 구조," 정보과학회논문지, VOL. 29, NO. 06 pp.0712~0721, 2002.12.

[5] 김영찬, "Operating System Concepts," 홍릉과학출판사, 1999.

[6] 이수영, 남택용, 나중찬, 손승원, "액티브 네트워크 기술 동향," ETRI 주간기술동향, 2001

[7] D.S Alexander, et al., "The SwitchWare Active Network Architecture," IEEE Network Specail Issue on Active and Controllable Networks, vol.12 no.3, 1998.