액티브 네트워크 환경에서 실행 코드 교체를 위한 ANC 캐싱 기법

정회원 장 창 복*, 이 무 훈**, 조 성 훈***, 최 의 인****

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

Chang-bok Jang*, Moo-Hun Lee**, Sung-Hoon Cho***, Eui-In Choi**** ***Reguler Member***

요 약

인터넷과 컴퓨터의 성능이 발달함에 따라 사용자들은 네트워크를 통해 많은 정보를 얻고 있다. 이에 따라 네트워크를 이용하는 사용자의 요구도 다양해지고 빠르게 증가하고 있다. 하지만 이러한 다양한 사용자 요구를 현재의 네트워크에서 수용하기에는 많은 시간이 걸리기 때문에, 액티브 네트워크와 같은 기술들이 연구되고 있다. 이런 액티브 네트워크 환경에서 액티브 노드는 이전 네트워크에서처럼 단순하게 패킷을 전달하는 기능뿐 만아니라 사용자의 실행 코드를 저장하고, 처리할 수 있는 기능을 가지고 있다. 따라서 액티브 노드에 전달된 패킷을 실행하기 위해서는 각 패킷을 처리하는데 필요한 실행 코드가 요구되고, 이러한 실행 코드가 실행하려는 액티브 노드 내에 존재하지 않을 경우 이전 액티브 노드나 코드 서버에 요청함으로써 얻을 수 있다. 하지만 이러한 실행 코드를 바로 액티브 노드에서 실행하지 않고, 이전 액티브 노드나 코드 서버에서 가져오게 되면 실행코드가 전달될 때까지의 시간지연이 발생하므로 네트워크의 트래픽 증가와 실행 시간 증가를 가져올 수 있다. 따라서 사용되었던 실행 코드를 액티브 노드의 캐시에 저장하여 코드의 실행 속도를 증가시키고 이전 액티브 노드로의 코드 요청 횟수를 감소시킬 필요가 있다. 따라서 본 논문에서는 액티브 노드 상에 실행 코드를 효율적으로 캐싱함으로써 실행 코드 요청의 횟수를 줄이고, 코드 실행 시간을 감소시킬 수 있는 ANC(Active Network Cache) 캐싱 기법을 제안하였다. 본 논문에서 제안한 캐싱 기법은 이전 노드로부터 실행 코드의 요청을 줄임으로써 코드의 실행시간을 단축시키고, 네트워크의 트래픽을 감소시킬 수 있다.

Key Words : Active Network, Cache policy, Code request

ABSTRACT

As developed Internet and Computer Capability, Many Users take the many information through the network. So requirement of User that use to network was rapidly increased and become various. But it spend much time to accept user requirement on current network, so studied such as Active network for solved it. This Active node on Active network have the capability that stored and processed execution code aside from capability of forwarding packet on current network. So required execution code for executed packet arrived in active node, if

* 한남대학교 컴퓨터공학과 데이터베이스 연구실(chbjang@dblab.hannam.ac.kr),
** 한남대학교 컴퓨터공학과 데이터베이스 연구실(mhlee@dblab.hannam.ac.kr),
*** 한남대학교 컴퓨터공학과 데이터베이스 연구실(shcho@dblab.hannam.ac.kr),
**** 한남대학교 컴퓨터공학과 데이터베이스 연구실(eichoi@dblab.hannam.ac.kr)
논문번호 : KICS2005-03-099, 접수일자 : 2005년 3월 8일
※ 본 연구는 산업자원부 지역협력연구사업(R12-2003-004-03002-0) 지원으로 수행되었음

execution code should not be in active node, have to take by request previous Action node and Code Server to it. But if this execution code take from previous active node and Code Server, bring to time delay by transport execution code and increased traffic of network and execution time. So, As used execution code stored in cache on active node, it need to increase execution time and decreased number of request. So, our paper suggest ANC caching technique that able to decrease number of execution code request and time of execution code by efficiently store execution code to active node. ANC caching technique may decrease the network traffic and execution time of code, to decrease request of execution code from previous active node.

Ⅰ. 서 론

네트워크와 컴퓨터의 성능이 발달함에 따라 사용자들은 네트워크를 이용하여 많은 정보를 얻고 있다. 이에 따라 네트워크를 통해 인터넷을 이용하는 사용자들의 요구도 점점 다양해지고 복잡해지고 있다. 하지만 현재의 네트워크는 새로운 기술이나 표준을 네트워크 망에 적용하기 위해서는 표준안 제정과 장비 교체와 같은 작업들로 인하여 많은 시일과 비용이 소요되고, 빠르게 변화하는 사용자 요구조건을 지원하기에는 네트워크 시스템의 변화 속도가 상대적으로 느리기 때문에 사용자의 네트워크에 대한 요구사항을 시기적절하게 반영하는 것이 사실상 불가능하다. 따라서 이러한 문제점을 해결하기 위해서 액티브 네트워크(active network)와 같은 연구가 활발하게 이루어지고 있다. 액티브 네트워크는 라우터나 스위치가 프로그램을 실행할 수 있는 능력을 가지고 있어서 프로그램을 포함하고 있는 패킷이나 중간 노드(또는 액티브 노드)에서 프로그램을 실행하도록 하는 패킷(또는 액티브 패킷)을 다양하고 유동적으로 처리할 수 있는 환경이다[1, 19, 25, 26]. 이러한 액티브 네트워크에 관한 연구로는 ANTS, Switchware, PAN, CANE, FAIN과 같은 것들이 있다[6, 7, 13, 20, 21]. 또한 기존 네트워크에서 패킷을 전달할 때, 단순히 중간 노드에서 패킷의 경로를 설정하고 전달했던 것과는 다르게 액티브 네트워크의 중간 노드에서는 다양한 작업을 처리할 수 있도록 하여, 기존의 네트워크에서 제공하지 못했던 유연성과 다양한 장점을 제공할 수 있다[2, 10, 13].

이러한 액티브 네트워크 상에서의 중간 노드는 패킷이 도착하면 패킷을 분류하고 패킷 내에 포함된 코드를 해석하여 실행 여부를 판단하며 패킷이 실행되면 그 결과를 다음 노드에 전달한다. 따라서 전달된 패킷을 처리하기 위해서는 처리할 데이터와 실행 코드가 필요하며 실행하기 위한 코드는 액티브 노드에 존재하여야 한다[23]. 만약 노드에 패킷을 실행하기 위한 실행코드가 존재하지 않으면, 이전 노드나 코드 서버로부터 실행 코드를 요청하여 전달 받아야 하기 때문에 네트워크의 트래픽이 증가되고 코드가 실행될 노드에 전달되기까지 시간상의 지연이 발생하여 실행시간의 증가를 가져오게 된다. 그러므로 실행 코드를 노드내의 캐시에 저장하여 실행 지연시간을 줄일 수 있는 캐시 기법에 관한 연구가 필요하다. 따라서 본 논문에서는 실행시간과 코드 요청 횟수를 줄이기 위해 참조 횟수와 시간 제약(Time limit)을 통한 효율적 ANC 캐시 기법을 제안한다.

2장은 관련 연구로써 액티브 네트워크와 기존 캐시 기법에 대하여 설명하고, 3장은 제안한 ANC 캐시 기법을 설명한다. 4장에서는 제안한 캐시 기법과 기존 캐시 기법의 성능을 비교 분석하고, 5장에서 결론 및 향후 연구 과제를 제시한다.

Ⅱ. 관련 연구

2.1 액티브 네트워크

2.1.1 액티브 네트워크 구조

액티브 네트워크의 중간 노드인 스위치나 라우터는 단순한 패킷 전달(forwarding) 기능 뿐만 아니라 사용자의 실행 코드를 저장/처리/전달할 수 있다. 사용자는 프로그램 코드를 패킷에 포함한 뒤 목적지로 전송하여 액티브 노드에서 실행하거나 미리 설치된 프로그램을 호출함으로써 자신이 원하는 네트워크 기능을 이용할 수 있다[1, 2, 9, 10, 12, 16, 18].

액티브 네트워크는 액티브 노드와 액티브 서버로 이루어지며, 액티브 노드는 기존 네트워크의 패킷에 해당하는 캡슐과 이를 중간에서 처리하는 기능을 담당하고, 액티브 서버는 캡슐을 생성하는 기능을 가지고 있다. 또한 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 전달하며, 노드 운영 체제, 실행환경, 액티브 응용(Active Application :

AA)으로 구성되어져 있다[15, 17, 22]. 노드 운영체제는 액티브 패킷을 전달 받아 이를 적절한 실행환경으로 분배하고, 실행환경에서 해당 액티브 패킷을 처리하기 위해 요구되는 자원을 관리하고 계산, 저장과 같은 요구를 중재하며, 실행환경으로부터의 결과를 다른 노드로 전달하는 기능을 수행한다. 실행 환경은 프로그램하거나 직접적으로 패킷을 제어할 수 있는 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. 따라서 실행 환경은 일반 범용 컴퓨팅 시스템에서의 쉘 프로그램처럼 동작하고, 접근할 수 있는 종단 간 네트워크 서비스를 통해 인터페이스를 제공한다. 이때 실행에 필요한 실행코드는 노드에 적재 되어 있어야 하며 만약 필요한 실행 코드가 노드에 존재하지 않을 경우 다른 노드로부터 전달 받아야 한다. 이러한 경우에 요구되는 기술이 코드 요청 기술로써, 현재 이전 액티브 노드로부터 전달 받는 방법과 코드 서버로부터 전달 받는 방법이 연구되고 있다[4].

2.1.2 코드 요청 기술

코드 요청 기술은 이전 액티브 노드로부터 전달 받는 방법과 특정 코드 서버로부터 전달 받는 방법이 연구되고 있으며, 그림 1은 ANTS의 코드 분배 프로토콜을 나타낸 것으로 액티브 노드에서 패킷을 처리하기 위해 실행 코드를 어떻게 요청하는지를 보여주고 있다[3]. 액티브 노드에서 패킷을 요청하고 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 일정시간이 지나도 요청에 대한 응답이 도착하지 않거나, 응답이 필요한 것이 아니라면 캡슐은 버려지게 된다.

따라서 액티브 노드 내에 실행 코드가 존재하지 않으면, 다른 노드로부터 코드를 요청해야 하기 때문에, 코드가 다른 노드로부터 전송되어지는데 까지 패킷이 처리되지 않고 대기 상태로 존재한다. 그러므로 좀더 빠른 코드 실행과 실행코드 요청 횟수의 감소를 위해서 실행될 코드를 액티브 노드의 캐시 내에 효율적으로 저장하여 교체할 수 있는 캐시 기법이 필요하다.

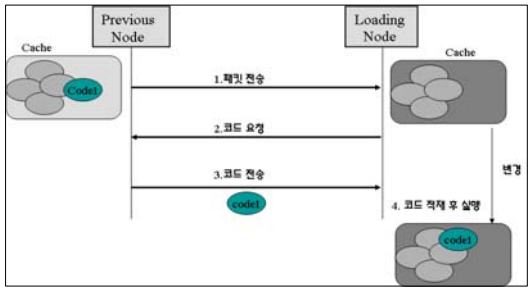

그림 1. ANTS의 코드 분배 프로토콜

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, 24]. FIFO 기법은 교체될 대상을 선택함에 있어 가장 먼저 저장된 것을 선택하여 교체하는 기법이다. 이 기법은 구현하거나 이해하기 쉽지만, 실행 코드를 캐시에 저장할 때 적재 공간이 부족하여 코드 중에 하나를 삭제하려 할 경우, 사용 빈도가 높아 다음에 사용될 가능성이 많음에도 불구하고 가장 먼저 적재된 코드라면 삭제되어야하는 문제점이 존재한다. LFU 기법은 데이터가 얼마나 많이 참조되었는지에 관한 정보를 저장하여 참조 횟수가 가장 적은 것을 교체한다. 따라서 자주 사용되는 코드를 계속 유지함으로써 좋은 성능을 나타낼 수 있지만 단기간에 많이 사용되고 오랜 기간동안 사용되지 않을 경우에, 참조 횟수가 높기 때문에 여전히 캐시에 저장되어 공간의 효율성을 떨어지게 하는 단점이 있다. LRU 기법은 최근에 참조되었는지에 관한 시간 정보를 이용하여 가장 오랫동안 참조되지 않은 것을 교체하는 기법으로, 오랫동안 참조되지 않는 데이터를 삭제할 수 있지만 참조 횟수를 반영하지 못하는 단점을 가지고 있다.

Ⅲ. 제안한 ANC(Active Network Cache) 캐시 기법

액티브 네트워크에서는 다양한 실행 코드가 존재

하며, 이러한 실행 코드는 액티브 노드에서 필요에 따라 실행된다. 이러한 실행 코드가 노드 내에 존재하지 않으면 이전 액티브 노드에 요청하여 실행코드를 전송받아야 하기 때문에 실행시간 지연과 네트워크 트래픽 증가를 가져오게 된다. 따라서 실행코드를 효율적으로 캐시에 유지하면, 코드 요청 횟수를 줄일 수 있고 패킷 처리 대기 시간의 감소와 패킷 처리 속도 증가를 가져올 수 있다.

본 논문에서는 이러한 실행코드를 좀더 효율적으로 캐시하기 위해 실행 코드가 얼마만큼 참조되었는지에 관한 참조 횟수와 최근에 참조되었는지를 판단하기 위한 시간 제약(time limit)에 대한 값을 코드 정보 테이블(Code Information Table)에 저장하고 관리하여 효율적으로 실행코드를 캐시에서 교체할 수 있도록 하였다. 이러한 코드 정보 테이블은 표 1과 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.

표 1. 코드 정보 테이블의 구성

Code ID	Execution NO	Time Limit No
A	5	1
B	1	1
C	7	0
D	8	8
E	7	7

식별자 필드는 실행할 코드를 식별하기 위한 것으로 코드의 고유 식별자가 저장되게 된다. 코드 실행 횟수 필드는 코드가 얼마만큼 실행되었는지를 저장하는 부분으로 코드가 최초 실행될 때 1 값을 가지며, 재실행 될 때마다 값이 1씩 증가된다. 시간 제약 필드는 최초 사용된 이후로 얼마만큼 오랫동안 사용되지 않았는지를 저장하는 부분으로 일정한 시간이 지나면 값을 갱신하여 유지한다. 즉, 코드가 최초로 실행된 후에 코드는 시간 제약에 대한 일정한 값(10)을 가지게 되고, 캐시 관리자는 코드가 실행된 후 일정 시간 마다(320초) 시간 제약 값을 1씩 감소시킨다. 시간 제약 값은 코드가 재실행 될 때 다시 초기 값(10)을 가진다. 캐시하기 위한 기억 공간이 한정되어져 있기 때문에, 캐시에 저장되어져 있는 실행 코드를 삭제하고 새로운 실행 코드를 적재하기 위해 본 논문에서는 다음과 같은 두 가지 방법을 적용하는 ANC 캐시 기법을 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 0인 실행 코드가 하나인 경우, 실행 횟수를 이용하지 않고 시간 제약 값이 0인 그 코드를 삭제하는 기법이다. 이 기법은 단기간에 참조되고 오랫동안 참조되지 않는 실행코드들을 삭제하여 캐시 공간의 효율성을 증가시킨다. 만일 시간 제약 값이 0인 경우가 한 개 이상일 때는 위의 첫 번째 방법을 적용한다. 그 외 시간 제약 값이 0인 경우가 두 개 이상일 경우는 실행 횟수가 가장 적은 코드를 삭제한다. 그림 2는 본 논문에서 제안한 ANC 기법의 캐시 교체 순서를 나타낸 것이며, 패킷 실행에 필요한 코드 요청에 대한 응답을 받은 상황부터 시작한다. 제안한 기법의 동작 과정은 다음과 같다.

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 값이 0인 경우를 검사한다.
③ 검사된 코드 중에서 시간 제약 값이 0인 경우가 하나 일 때는, 그 경우의 실행 코드를 삭제한다.
④ 검사된 코드 중에서 시간 제약 값이 0인 경우가 0개 이거나 한 개 보다 많은 때에는, 실행 횟수가 가장 적은 코드를 삭제한다.
⑤ 캐시 관리자는 코드 적재를 위해 필요한 공간이 생길 때 까지 위의 과정을 반복한다.

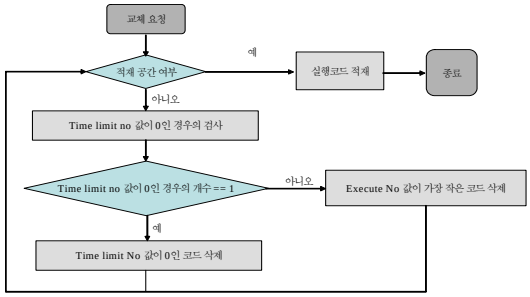

그림 2. 제안한 ANC 기법의 동작 과정

예를 들어 표 1의 코드 정보 테이블처럼 실행 코드 A, B, C, D, E가 캐시 내에 저장되어 있고, 실행 코드 F를 캐시에 적재하려고 할 경우, 코드 적재 공간이 부족하기 때문에 실행코드를 삭제해야 한다. 따라서 A, B, C, D, E 코드 중에서 시간 제약 값이 0인 경우가 하나이기 때문에 본 논문에서 제안한 두 번째 기법에 의해 C 코드를 삭제하고 F 코드가 적재된다. 만일 320 초가 지난 후(시간 제약 값이 1 감소) 새로운 코드 G가 캐시에 적재될 경우, 시간 제약 값이 0인 코드가 A, B 이기 때문에 (320초 동안 어떠한 코드도 액티브 노드 상에서 실행이 되지 않았다고 가정), 본 논문에서 제안한 첫 번째 기법에 의해 삭제될 코드는 실행 횟수가 가장 작은 B 코드가 된다. 따라서 최종 코드 정보 테이블은 표 2와 같이 된다.

표 2. 실행 코드 삭제 이후의 코드 정보 테이블

Code ID	Execution NO	Time Limit No
A	5	0
G	1	10
F	1	9
D	8	7
E	7	6

본 논문에서 제안한 기법에서는 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 좀 더 효율적으로 사용할 수 있도록 하였다.

Ⅳ. 성능 분석

본 장에서는 제안한 ANC 캐시 기법의 성능을 평가하기 위한 시뮬레이션과 그 결과에 대하여 설명한다. 시뮬레이션은 제안된 기법을 통해 액티브 노드에서 코드 요청 횟수의 감소를 통한 네트워크 트래픽 감소 여부와 빠른 실행의 가능성 여부에 중점을 두고 실행하였다. 그리고 코드 요청 횟수와 코드 실행 횟수 그리고 코드 실행 시간을 시뮬레이션의 매개 변수로 설정하고 실험하였다.

4.1 실험 환경

본 논문에서 제안한 ANC 캐시 기법의 수행 알고리즘은 AweSim 시뮬레이션 패키지 Version 2.0을 사용하여 수행하였고, 제안된 캐시 기법과 다른 기법은 AweSim에서 사용가능한 프로세스 중심(process oriented) 방식의 네트워크 모델과 제어 기법으로 표현하였다. AweSim 시뮬레이션 패키지는 자체의 처리기와 클록을 가지고 있기 때문에 실험 시스템의 속도에는 영향을 받지 않는다[8]. 실험은 Pentium-IV 2.4GHz, Windows 2000 Advanced Server 환경에서 수행하였고, 성능 분석 모델은 본 논문에서 제안한 캐시 기법을 기초로 하여 코드 실행에 따른 코드 요청 횟수로 시뮬레이션 환경을 설정하고 코드의 중복률과 시간 제약 값에 변화를 주어 실행하였다.

4.2 실험 전략 및 가정

본 논문에서 제안한 캐시 기법은 실행할 코드가 캐시 내에 존재하는지 확인한 다음 존재하지 않을 경우, 이전 노드로 코드를 요청한다. 이 때, 코드 요청시 이전 노드로부터 코드가 획득되지 않는 경우는 실험에서 배제하였으며, 이전 노드로부터 코드가 전송될 때 일정한 전송 속도를 갖도록 하였다. 또한 실행 코드는 최대 5개까지 캐시 내에 저장할 수 있도록 하였으며, 액티브 노드 내 코드 실행 시간은 일정한 범위 내에서 랜덤 실행 값을 갖도록 하였다. 그리고 실행 코드는 같은 크기를 갖는 것으로 가정하여, 서로 다른 실행 코드의 크기에 따른 코드 적재 방법에 대해서는 고려하지 않았다.

시뮬레이션에서의 평가 대상과 항목은 다음과 같다.

1) 평가대상
   FIFO, LRU, LFU, ANC 기법
2) 평가항목
   코드 요청 횟수, 코드 실행 횟수, 코드 실행 시간

4.3 성능 분석 결과

4.3.1 실행 코드의 중복률 1에 대한 분석

본 실험에서는 코드의 중복률에 따라 각 캐시 기법의 코드 실행 횟수, 코드 요청 횟수, 실행시간을 측정하기 위해서 중복률 1, 2, 3, 5, 10, 15 순서로 중복률을 정의하였다. 중복률은 실행 코드의 종류가 얼마만큼 중복되어져 있는가를 나타내는 것으로 숫자가 커질수록 중복의 횟수가 적어지고 다양한 코드들이 존재하게 된다. 즉 중복률 1일 경우에는 같은 종류의 코드들이 매우 많이 존재하게 되고, 캐시 적중률(hit ratio)이 높아지게 된다.

먼저 실행하는 코드의 중복이 많을 경우(중복률 1)에 코드 요청 횟수가 현저히 줄어들기 때문에 본 논문에서 제안한 ANC 기법과 LRU, LFU, FIFO 기법과의 성능차이가 거의 없었다. 다음 그림 3, 그림 4, 그림 5는 각 캐시 기법들을 코드 중복이 높을 경우의 코드 실행 횟수, 코드 요청 횟수, 코드 실행 시간을 비교한 그림이다. 그림에서 코드 실행 횟수는 실제 코드가 실행한 개수가 아닌 이전 노드로 코드를 요청하지 않고 바로 액티브 노드 내에서 실행된 코드수를 총 코드 수로 나눈 후 백분율로 환산한 값이다.

그림 3에서 중복이 높은 경우 FIFO 기법이 가장 적은 실행 횟수를 나타냈으며, 그 외 3가지 기법은 거의 유사한 실행 횟수를 보이고 있다. LFU 기법인 경우 전체 실행 코드 중 94.9%에 해당하는 코드가 이전 노드로 요청되지 않고 바로 실행되었다. 그림 4는 각 기법의 실행시간을 비교한 것으로 거의 유사한 것을 볼 수 있다. 또한 본 실험에서 실행시간을 계산할 때 일정한 범위에서 랜덤하게 값을 갖도록 해주었기 때문에 랜덤 값에 따라 오차범위 ±1.0초를 갖는다.

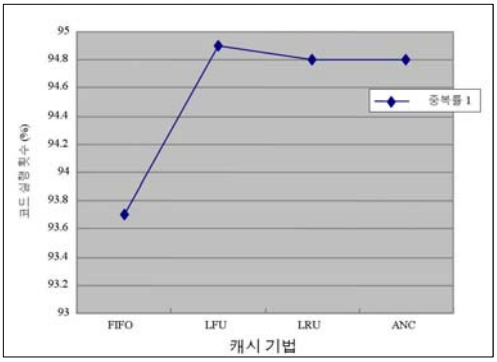

그림 3. 중복률 1일 때의 코드 실행 횟수

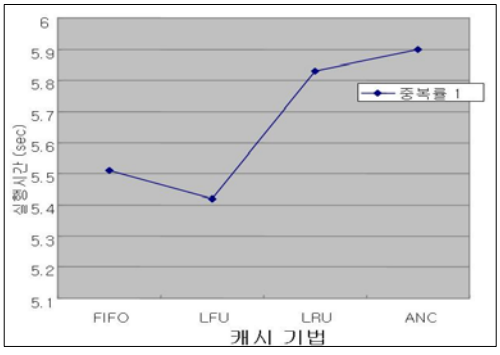

그림 4. 중복률 1일 때의 실행 시간

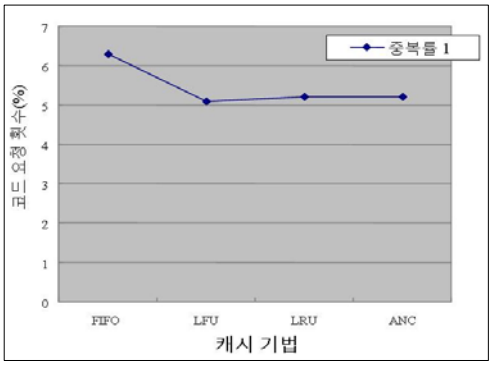

그림 5. 중복률 1일 때의 코드 요청 횟수

그림 5의 코드 요청 횟수에 따른 비교에서도 FIFO 기법을 제외한 3가지의 기법이 거의 유사한 성능을 갖는 것을 볼 수 있다. LFU 기법이 전체 실행 코드 중 5.1% 정도의 코드만이 이전 노드로 실행 코드를 요청하였다.

4.3.2 서로 다른 중복률에 따른 분석

본 실험에서는 중복률 1을 제외한 중복률 2, 3, 5, 10, 15에 대해 시간 제약 값으로써 논문에서 제안한 ANC 캐시 기법은 320초인 경우(ANC-320)와 80(ANC-80)초인 경우를 적용하였고, LRU 기법은 200초인 경우(LRU-200)만을 적용하여 실험하였다. 본 실험에서 ANC와 LRU의 시간 제약값이 다른 이유는 LRU의 경우 시간 제약값이 200초인 경우가 가장 좋은 결과를 나타냈기 때문에 본 논문에서는 두 가지의 경우만 비교 분석하였다. 또한 ANC역시 두 가지의 시간 제약 값(320초, 80초)을 갖도록 하여 시간 제약 값이 서로 다를 경우의 성능 차이를 비교하였다. 각 기법에 대한 성능을 평가한 결과는 아래와 같다.

그림 6은 각 캐시 기법에 따라 중복률이 2인 경우에서 15인 경우까지 얼마나 많은 코드가 바로 실행되었는지를 보여주고 있다. 그림 6을 통하여 중복률이 적어질수록 다양한 실행코드가 존재해지기 때문에 코드의 실행 횟수가 적어지는 것을 볼 수 있으며, FIFO 알고리즘이 가장 낮은 코드 실행 횟수를 보여주고 있는 것을 확인할 수 있다. 그리고 LFU와 ANC-320 기법이 중복률에 상관없이 다른 기법에 비해 좋은 성능을 가지는 것을 볼 수 있다. 하지만 ANC-80일 경우는 적절치 못한 시간 제약 값으로 인해 LFU와 ANC-320보다 오히려 낮은 성능을 갖는 것을 볼 수 있다. 따라서 적절치 못한 시간 제약 값은 오히려 자주 사용되는 코드를 삭제시킬 수 있는 확률이 증가되어 성능 감소의 원인이 된다.

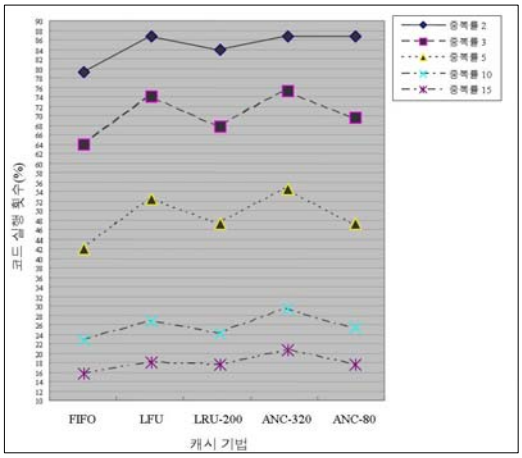

그림 6. 각 캐시 기법에서의 코드 실행 횟수

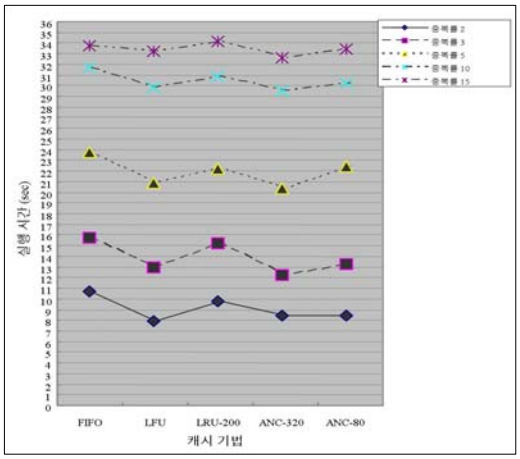

그림 7. 각 캐시 기법에서의 실행 시간

그림 7은 중복률에 따른 각 캐시 기법별 실행 시간을 보여주고 있다. 역시 ANC-320과 LFU가 보다 빠른 실행 시간을 가지고 있으며, FIFO 알고리즘과 LRU 알고리즘이 가장 느린 실행시간은 가지고 있다. 그리고 중복률이 적어질수록 코드의 요청이 많아지기 때문에 점차적으로 실행시간이 증가되는 것을 볼 수 있다.

아래의 그림 8은 본 실험에서 가장 좋은 성능을 보였던 ANC 기법과 LFU 기법의 중복률에 따른 코드 실행 횟수만을 비교 분석한 그림이다. 중복률 가장 높은 경우는 거의 유사한 성능을 가진 반면에, 그 외의 중복률에서는 본 논문에서 제안한 ANC-320 기법이 보다 높은 코드 실행 횟수를 갖고 있는 것을 볼 수 있다. 하지만 ANC-80은 잘못된 시간 제약 값으로 LFU보다 오히려 낮은 코드 실행 횟수를 보이고 있다.

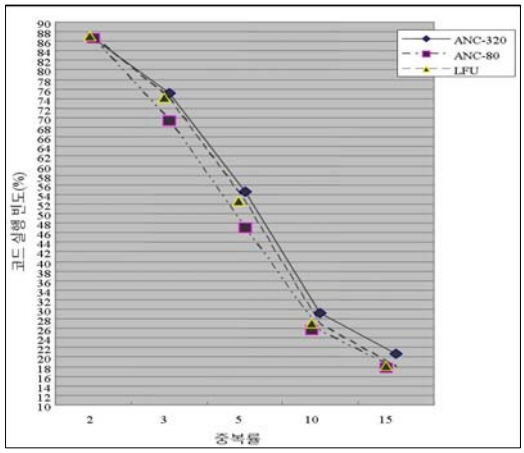

그림 8. LFU와 ANC-320, ANC-80 기법의 비교

V. 결 론

본 논문에서는 액티브 노드 상에서 동적으로 패킷을 처리하고, 실행 코드 요청 횟수를 줄이기 위해 액티브 노드의 캐시에 실행코드를 저장하여 사용할 수 있도록 하였다. 캐시 내에 저장된 실행 코드는 패킷의 요구에 따라 실행되며, 만일 캐시 내에 저장되어져 있지 않다면 이전 액티브 노드로부터 실행 코드를 요청한다. 따라서 본 논문에서는 제한된 캐시저장 용량에 좀 더 효율적으로 실행코드를 저장하여 실행코드 요청을 줄이기 위한 효율적인 캐시 교체 알고리즘으로 ANC를 제안하였다. 제안된 ANC 캐시 기법을 사용함으로써 액티브 노드 상에서 패킷 처리 대기 시간을 감소시키고 패킷 처리의 속도 향상을 기대할 수 있으며, 코드 요청 횟수와 캐시 공간 낭비를 줄일 수 있다. 또한 코드의 요청 횟수의 감소로 네트워크 트래픽을 감소시킬 수 있는 장점을 갖는다.

향후 연구 과제로 잘못된 시간 제약 값은 오히려 자주 사용되는 코드들을 삭제시켜 성능 저하를 가져올 수가 있기 때문에, 실행 코드의 중복률과 상관없이 가장 효율적인 교체를 위한 시간 제약 값에 관한 연구가 필요하다.

참 고 문 헌

[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, 1977.

[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," *정보과학회 논문지*, 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.

[8] Jack Jensen, "A Guide to Business Decision-Making Using Visual SLAM II and AweSim", 1999

[9] K. L. Calvert, "Architectural Framework for Active Networks Version 1.0", *Active Network Working Group*, DRAFT July 27, 1999

[10] Konstantinos Psounis, "Active Networks: Application, Security, Safety, And Architectures", *IEEE Communications Surveys*, 1999

[11] Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, et. al, "Smart Packets: Applying Active Networks to Network Management", *ACM Transactions on Computer Systems*, Vol. 18, No. 1, February 2000, Pages 67-88.

[12] Thomas Becker, et al, "Initial Active Network and Active Node Architecture ver 1.0", May 2001

[13] M. Hicks, et. al, "PLANet: An Active Internetwork", *IEEE INFOCOM*, 1999

[14] T. Wolf, et. al, "A Scalable High- Performance Active Network Node", *IEEE Network*, 1999

[15] S. Merugu, et. al., "Bowman: A node OS for Active Networks", *Proceedings of IEEE Infocom 2000*, March 2000

[16] G. Alex, et. al., "A Flexible IP Active Networks Architecture", *IWAN 2000 Conference*, 2000

[17] L. Peterson(Editor), "NodeOS Interface Specification", *DARPA AN NodeOS Working Group*, 1999

[18] D. Wetherall, et. al., "The Active IP Option", *7th ACM SIGOPS European Workshop*, 1996

[19] Danny Raz, Yuval Shavitt. "An Active Network Approach to Efficient Network management", *Proceedings of the First International Working Conference on Active Networks (IWAN '99)*, 1999

[20] Erik L. Nygren, Stephen J. Garland, and M. Frans Kaashoek, "PAN: A High-Performance Active Network Node Supporting Multiple Mobile Code Systems", *IN PROCEEDINGS IEEE OPENARCH'99*, MARCH 1999

[21] Jonathan T. Moore Scott M. Nettles. "Towards Practical Programmable Packets", Technical Report MS-CIS-00-12, Department of Computer and Information Science, University of Pennsylvania, May 2000 (switchware)

[22] S.Merugu S.Bhattacharjee et al. "Bowman and CANEs : Implementation of an Active Network". *In Proceedings of 37th Annual Allerton Conference*, Monticello, IL,September 1999

[23] Samrat Bhattacharjeey Martin W. McKinnon. "Performance of Application-Specific Buffering Schemes for Active Networks", Technical Report GIT-CC-98-17, College of Computing, Georgia Tech

[24] Ying Shi, Edward Watson, Ye-sho Chen. "Model-driven simulation of world-wide-web cache policy", *Proceedings of the 1997 Winter Simulation Conference*, 1997

[25] Tal Lavian, Phil Yonghui Wang. "Active Networking On A Programmable Networking

Platform", *IEEE OPENARCH 2001*, 2001
[26] Danny Raz, Yuval Shavitt. "Active Networks for Efficient Distributed Network management", *IEEE Communications Magazine*, March 2000

장 창 복(Chang-bok Jang) 정회원

2001년 2월 한남대학교 컴퓨터공학과 졸업
2003년 2월 한남대학교 컴퓨터공학과 석사
2003년 3월~현재 한남대학교 컴퓨터공학과 박사과정
<관심분야> 액티브 네트워크, 유비쿼터스, Middleware, Active network, Context Modeling, Semantic Web, Software Security, OVPN

이 무 훈(Moo-hun Lee) 정회원

2002년 8월 한남대학교 컴퓨터공학과 졸업
2004년 8월 한남대학교 컴퓨터공학과 석사
2004년 9월~현재 한남대학교 컴퓨터공학과 박사과정
<관심분야> Semantic Web, Web Service, Web search engine, Data Stream Management System

조 성 훈(Sung-hoon Cho) 정회원

2001년 2월 한남대학교 컴퓨터공학과 졸업
2003년 2월 한남대학교 컴퓨터공학과 석사
2004년 3월~현재 한남대학교 컴퓨터공학과 박사과정
<관심분야> RDF, Semantic Web, Ontology, ebXML, Web Service

최 의 인(Eui-in Choi) 정회원

1982년 2월 한남대학교 계산통계학과
1984년 2월 홍익대학교 전자계산학과 석사
1995년 2월 홍익대학교 전자계산학과 이학박사
1985년~1988년 공군 교육사 전산실장
1992년~1996년 명지전문대학 전자계산과 조교수
1996년 3월~현재 한남대학교 컴퓨터공학과 교수
2003년 UCLA visiting scholar
<관심분야> Web Service, Semantic Web, Web search engine, Ubiquitous Computing, Context-Aware Retrieval, Context Modeling, Data Stream Management System