SIP Proxy 를 위한 캐슁 기법 제안

박경수*, 박명순*
*고려대학교 컴퓨터과학 기술 대학원
e-mail : ablax@erasinfo.com, myongsp@korea.ac.kr

Proposed Caching Method for SIP Proxy

Kyung-soo Park*, Myong-soon Park*
*The graduate school of Computer Science & Technology, Korea University

요 약

SIP Proxy 서버는 SIP UA 가 전송하는 대량의 Call 메시지를 처리해야 한다. SIP Proxy 서버가 SIP UA 에 의해 전송된 Call Message 를 처리 하기 위해서는 Registrar 서버에 등록 되어 있는 UA 들의 정보를 빠르게 검색하여 처리해야 한다. 본 논문에서는 SIP Proxy 서버가 Registrar 서버에 등록 되어 있는 SIP UA 에 대한 접속 정보를 빠르게 처리하기 위해 SIP Proxy 서버와 Registrar 서버에 UA 정보용 cache 를 도입할 것을 제안 하며, SIP Proxy 서버와 SIP UA 의 상호 동작 패턴을 적용한 SIP UA 정보 캐슁 기법에 대해서 제안 한다.

1. 서론

2000 년 초 새롬데이타기술의 인터넷 폰의 소개는 VoIP(Voice Over IP)라는 기술을 세계에 널리 알리기에 부족함이 없었다. 새롬데이타기술에 의해 알려진 VoIP 는 H.323 프로토콜을 사용하여 다양한 서비스를 제공하며 기존의 PSTN 망의 서비스와 대응하기 위한 연구가 진행되었다. 그러나 이 때 사용된 H.323 프로토콜은 구성의 복잡성, 인터넷 환경에서의 확장성 제한 등과 같은 단점을 가지고 있었다. 이러한 H.323 프로토콜의 단점에 대한 대안으로 1999 년 3 월 멀티미디어 세션 연결 및 관리를 위한 프로토콜인 SIP(Session Initiation Protocol)가 소개 되어졌다 [1,2]. 이후 SIP 는 VoIP 를 위한 프로토콜로서 개발이 되어 왔다. 기존의 멀티미디어 세션의 연결 및 유지를 위해 사용되어지던 H.323 프로토콜의 단점을 극복하고 VoIP 의 새로운 세션 유지 프로토콜로서 SIP 는 다양한 기능의 프로토콜로서 발전되어져 왔다. 그러나 기존의 SIP 프로토콜 연구들은 VoIP 의 기능을 SIP 프로토콜에 심어주는 연구에 집중되어 실제 VoIP 가 상용화 되기 위한 기능적인 연구에선 많은 연구가 진행되었다. SIP 프로토콜이 H.323 프로토콜 보다 단순하며 응용하기 쉽다는 장점이 있기는 하지만 성능적인 효과가 입증되어 있지 않다. 따라서 SIP 프로토콜을 상용화 하기 위하여 SIP 프로토콜을 사용한 SIP 컴포넌트 들의 성능 개선 연구가 필요하다.

본 논문에서는 SIP UA(User Agent)에 의한 대량의 call 메시지들을 처리해야 하는 SIP proxy 서버의 성능 개선을 위한 SIP UA 정보용 캐슁 기법을 제안했다.

본 논문의 제 2 장에서는 기존 캐슁 기법에 대해서 설명하며 제 3 장에서는 SIP 프로토콜이 가지고 있는 SIP Proxy 와 SIP UA 의 동작 패턴 특징에 대해서 분석한다. 제 4 장에서는 본 논문에서 제안 하는 UA 별 캐슁 기법에 대해서 설명하며 제 5 장에서는 결론 및 향후 연구 과제를 서술한다.

2. 캐슁 기법

캐슁 기법은 저장 장치 계층 간의 속도 차이를 완충 시켜주기 위해 컴퓨터 구조, 운영 체제 데이터베이스 등의 분야에서 각각 캐쉬 메모리, 페이징 기법, 버터링 기법 등으로 널리 연구 되어 왔다.

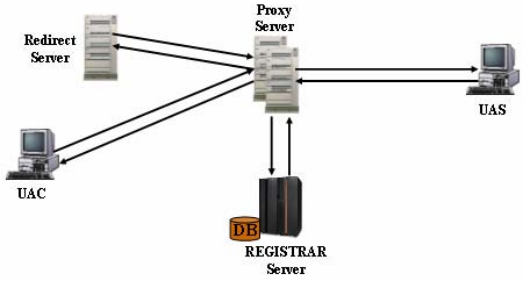
캐슁 기법의 성능은 캐쉬 교체 알고리즘에 의해 크게 좌우 된다. 캐쉬 교체 알고리즘은 미래의 참조를 미리 알지 못하는 상태에서 한정된 캐쉬 공간에 보관하고 있을 데이터와 삭제할 데이터를 동적으로 결정하는 온라인 알고리즘 이다.

효율적인 캐쉬 교체 알고리즘이 되기 위해서는 시간과 공간 복잡도 측면에서 현실성이 있어야 한다. 캐쉬 내에 있는 객체의 수가 n 개라 할 때 시간 복잡도 측면에서는 캐쉬 운영에 드는 시간이 $O(\log_2 n)$을 넘지 않는 것이 바람직하며 공간 복잡도 측면에서는

캐쉬 내의 객체 하나당 부가적으로 저장되는 정보가 상수, 즉 *O(1)*을 넘지 않는 것이 바람직한 것으로 알려져 있다.[4] 기존에 알려진 캐쉬 기법의 성능을 비교하였을 때 일반적으로 효율적인 성능을 발휘 하며 가장 많이 사용하는 기법은 LRU(Least Recently Used) 기법이다 그러나 발생 빈도 수를 중요시 하는 경우 확률 참조 위주의 LFU(Least Frequency Used) 기법이 적당하다.[7] 본 논문에서는 인터넷 사용자의 대부분이 인터넷 및 관련 응용의 사용에 있어 "최근 참조 성향" 보다는 "참조 빈도"에 더 높은 비율을 두고 있다는데 대해서[4,6,7] 발생 빈도에 따라 성능이 좀더 효율적인 LFU 기법을 기존 비교 기법으로 제안한다. 대부분의 기존 연구들이 효율적인 성능을 발휘 하는 것은 사실이다. 그러나 기존의 캐쉬 기법에 SIP 프로토콜의 동작 특징을 적용한다면 좀더 효율적인 캐쉬 기법이 될 것이다.

3. SIP 프로토콜의 동작 패턴 분석

SIP 프로토콜은 H.323 프로토콜 보다 단순하고 응용하기 쉽다라는 장점으로 최근 텔레포니 서비스, 인스턴트 메신저, 위치 통보 서비스, 이동성 지원서비스 등 다양한 연구에 사용되고 있다.[5] SIP UA 와 SIP Proxy 서버의 동작 패턴은 SIP Proxy 서버의 캐쉬 정책을 결정하는데 매우 중요한 요소가 된다.



(그림 1) SIP Protocol Architecture

UA 는 SIP Proxy 서버에게 연결을 원하는 목적 UA 에 대한 Call 메시지를 전송한다. SIP Proxy 서버는 해당 call 메시지의 내용을 분석하여 목적 UA 에게 해당 call 메시지를 전달 한다. 이때 Proxy 서버는 Registrar 서버가 관리하는 DB 에서 목적 UA 의 IP 정보 및 연결 포트 정보와 같은 연결 정보를 검색하여 해당 IP 로 call 메시지를 전달한다. 이때 UA 와 Proxy 서버의 동작패턴은 인스턴트 메신저나 혹은 휴대폰의 동작 패턴과 유사한 패턴으로 동작을 한다. 또한 UA 는 목적 UA 에 대한 정보를 사용자가 편리 하도록 주소록의 형태로 제공하며 UA 의 사용자는 주소록에서 간단하게 목적 UA 의 연결 정보를 알아내어 Proxy 서버에게 call 연결을 요청하게 된다.[1,2,6]

SIP UA 및 Proxy 서버 동작 패턴 정리

1. UA 는 Proxy 서버를 통해 목적 UA 와 통신을 한다.
2. UA 는 목적 UA 의 대표 ID 리스트를 보유 하며 UA 클라이언트는 주소록 기능을 제공한다.
3. Proxy 서버는 UA 에서 요청된 Call 메시지를 DB 내의 User 정보를 기반으로 목적 UA 에 전달한다.
4. UA 는 자주 통신하는 ID 를 주소록에 저장하고 자주 통신하는 ID 를 기억하고 사용한다.

위와 같은 동작 패턴은 SIP 프로토콜의 다른 응용 연구에 적용 되었을 때 기본적으로 제공 되는 기능을 바탕으로 한 동작 패턴을 정리한 것이다. SIP Proxy 서버가 SIP UA 의 call 메시지를 효율적으로 처리하기 위해 SIP 시스템에 캐쉬가 도입 되었을 때 이와 같은 SIP 프로토콜의 특수한 동작 패턴에 대한 특징이 캐쉬 정책에 도입 되어야 한다.

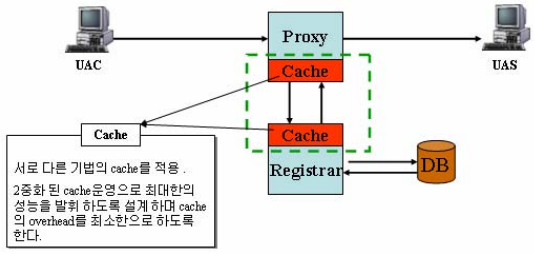
4. 개별 캐쉬 기법

논문에서는 제 3 장에서 설명한 SIP 프로토콜의 동작 패턴 분석을 통해 SIP 시스템의 동작 패턴 특징을을 발견하고 SIP 시스템에 SIP UA 와 SIP Proxy 서버 간의 동작 패턴 특징에 적합한 캐쉬 기법을 적용한 개별 캐쉬 기법인 UA 정보 캐쉬를 제안하고자 한다.

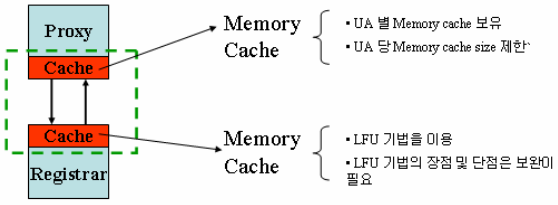
4.1 제안 개별 캐쉬 기법

4.1 절에서 설명한 바와 같이 UA 의 주소록의 연결 정보에 대한 내용을 Proxy 서버와 Registrar 서버에 캐쉬 형태로 저장하며 Proxy 서버의 캐쉬는 사용자별로 캐쉬를 관리하는 기법을 사용하고 Registrar 의 캐쉬는 Proxy 서버의 캐쉬에서 포함하지 못하는 나머지 정보를 일정 정도 캐쉬 한다.

제안 시스템은 Proxy 서버와 Registrar 서버의 이중화된 캐쉬를 사용하며 Proxy 서버의 캐쉬에 보관되지 못한 정보 중 일정 부분의 정보를 Registrar 서버에 존재하는 캐쉬를 사용하여 캐쉬 서비스를 제공하게 된다.



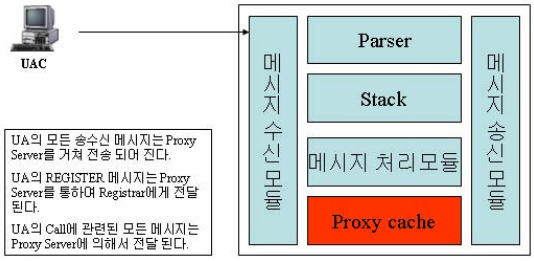
(그림 2) 제안 시스템 캐쉬 구조



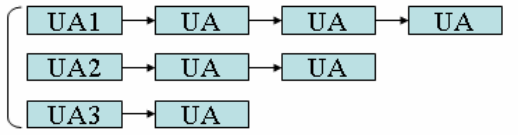
(그림 3) 시스템 캐쉬 구성

Proxy 서버는 UA 에 의해서 요청되는 Call 메시지를 처리하기 위해서 DB 에 저장 되어 있는 목적 UA 의 접속 정보를 검색한다. 해당 목적 UA 의 접속 정보를 검색하고 나면 call 메시지의 목적 UA 의 ID 부분을 접

속 정보로 처리한 후 해당 목적 UA에 전송하게 된다.
Proxy 서버는 Registrar DB에 저장되어 있는 UA 정보를 캐쉬에 저장하게 된다. 여기서 Proxy 서버가 관리하는 캐쉬는 UA 별로 관리 되며 UA 별로 제한된 캐쉬 크기를 적용하게 된다. 또한 Proxy에 의해서 관리되는 UA 별 캐쉬의 수량 또한 제한되어 관리 된다. Proxy 서버는 일정수준의 사용을 하는 UA를 대상으로 하여 UA별 주소록에 해당하는 캐쉬를 관리하며 UA 정보 캐쉬에는 일정 크기의 목적 UA 의 정보가 캐쉬 된다.

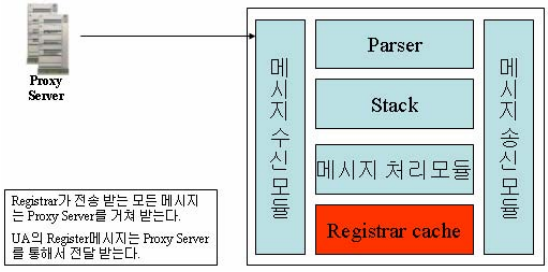


(그림 4) Proxy 서버 UA 캐쉬 모듈



(그림 5) Proxy 서버 캐쉬 개념도

Registrar 는 Proxy 서버의 개인별 캐쉬에 의해 캐쉬되지 못하는 목적 UA 의 정보를 캐쉬한다. Registrar 서버에 의해 관리되는 캐쉬는 관리에 대한 부하가 심하지 않고 구현이 비교적 단순한 LFU 기법을 이용한다. LFU 기법을 registrar 에 사용하는 이유는 복잡도에 비해서 비교적 많은 양을 캐쉬 할 수 있으며 일반적인 효율성에서 봤을 때 특정 사용자 정보인 proxy 서버의 UA 별 캐쉬를 보안 하기 위해서는 proxy 서버의 캐쉬를 보안하기 충분한 분량을 소화 하면서도 다른 캐쉬의 성능을 저하하지 않기 때문이다. 또한 SIP 시스템의 특징을 적용하지 않은 기존의 시스템을 대표하기 때문이다[5].



(그림 6) Registrar 서버 캐쉬 모듈

Registrar 서버의 캐쉬는 Proxy 서버 캐쉬의 수퍼 셋으로 Registrar 서버의 캐쉬에는 Proxy 서버의 캐쉬 정보가 포함 되어져 있다.
따라서 UA 의 call 메시지를 처리하고 전달해야 하는 기능을 가지고 있는 Proxy 서버에 복잡한 캐쉬를 크게 두지 않고 일정부분만 할당 하여 캐쉬를 관리하는데 불필요하게 부하를 가져 가지 않도록 하고 이에 따라 약해진 캐쉬의 기능을 Registrar 서버가 관리하는 캐쉬가 Proxy 서버의 캐쉬의 약점을 보안 하도록 한다.

4.2 성능 분석

제안 시스템의 Proxy 의 개별 캐쉬와 Registrar 의 수퍼 셋 캐쉬를 적용 하므로 해서 해당 캐쉬 시스템을 적용한 시스템은 아무런 캐쉬 정책을 적용하지 않은 시스템 보다 DB 검색 하는 시간만큼의 빠른 처리 속도를 유지 한다. 각각의 UA 의 call 메시지를 처리하는데 다음과 같은 수식으로 표현 할 수 있다.

T : call 을 처리 하는데 걸리는 시간
Tp : call 메시지의 스택을 처리하는데 걸리는 시간
Tsdb : call 메시지 내의 목적 UA 의 정보를 검색하는데 걸리는 시간(캐쉬를 사용하지 않는 DB 검색)
Tsm : LFU 기법을 적용한 캐쉬 내의 UA 정보를 검색하는데 걸리는 시간
Tsm_n : 각각의 캐쉬 메모리를 검색하는데 걸리는 시간
Tsc : 제안 개별 캐쉬 메모리를 검색하는데 걸리는 시간
Tscp : 제안 proxy 서버 캐쉬를 검색하는데 걸리는 시간
Tscr : 제안 registrar 서버 캐쉬를 검색하는데 걸리는 시간
** 단 제안 개별 캐쉬 시스템에서는 네트워크 통신에 대한 시간은 고려하지 않는다.*

$$T = Tp + Tsdb$$
$$T = Tp + Tsd + Tss$$
(일반 캐쉬를 적용하지 않은 시스템)

$$T = Tp + Tsm + Tsdb$$
$$Tsm = Tsm_1 + Tsm_2 + \cdots + Tsm_n$$
(LFU 기법을 적용한 캐쉬)

$$T = Tp + Tsc + Tsdb$$
$$Tsc = Tscp + Tscr$$
$$Tsc = (Tscp1 + \cdots + Tscpmax) + (Tscr1 + \cdots + Tscrn)$$
(제안된 기법을 적용한 캐쉬)

이상의 수식을 볼 때 제안된 캐쉬 기법을 적용한 시스템의 경우 제한된 개인별 캐쉬는 hit rate 에 따른 UA 정보를 검색하는데 소요되는 시간을

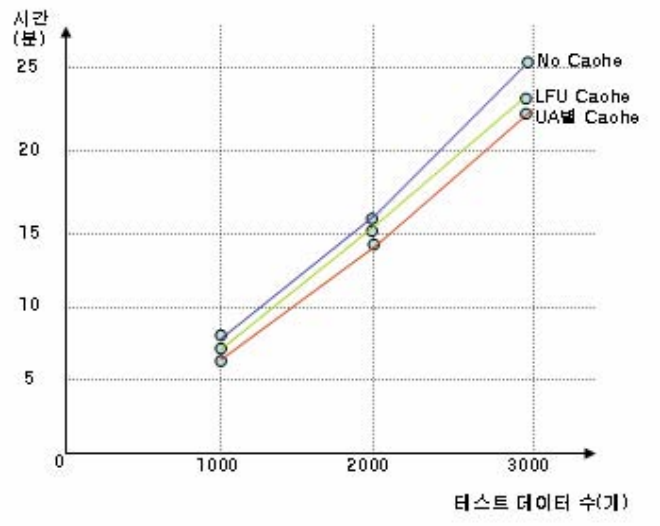
Tsc((전체 캐쉬 사이즈) - (UA 별 캐쉬 사이즈))

만큼 줄이는 역할을 하게 되므로 기존의 캐쉬 기법 보다 그만큼의 목적 UA 의 접속 정보 검색 시간을 줄이므로 해서 처리효율을 향상 시킬 수 있다.

시뮬레이션 환경
CPU : 인텔 펜티엄 3 850MHz
메모리 : 576 MB
OS : windows 2000 Professional
사용툴 : visual studio 6.0
DBMS : mysql 4.0.12-nt(no index)

데이터수 : 1000,000 개 중
1,000 개, 2,000 개, 3,000 개
registrar 캐쉬 : 데이터 수의 각 10%
proxy 캐쉬 : registrar 캐쉬의 10%

비교 캐쉬 기법
1. 기법을 적용하지 않음
2. 기존의 SIP 의 특징을 적용하지 않은 LFU 기법
3. UA 별 Cache 기법



(그림 7) 성능 분석 결과표

위의 성능분석 그래프는 단위 테스트 데이터 개수에 대해 각각의 캐쉬에 대한 검색 및 캐쉬 교체 시간을 측정한 것이다. 이때 데이터 베이스는 각각의 캐쉬의 성능 비교를 뚜렷하게 하기 위하여 인덱스를 사용하지 않았다. 또한 No cache 그래프는 SIP 시스템에 캐쉬를 도입하지 않은 상태의 데이터 베이스를 직접 액세스 하는 것을 의미 하며, LFU cache 는 SIP 시스템의 동작 특징을 적용하지 않은 기존의 cache 시스템을 대표 한다. UA 별 cache 는 본 논문에서 제안하는 proxy cache 와 registrar cache 의 이중화된 캐쉬에 대한 성능 테스트 그래프를 의미한다.

성능 시뮬레이션은 데이터 수의 10%크기를 registrar 의 캐쉬인 LFU 캐쉬로 사용하고, LFU 캐쉬의 10%를 개별 캐쉬로 설정하고 시뮬레이션을 하였다. 성능 시뮬레이션 결과 캐쉬를 전혀 사용하지 않을 때보다 SIP 의 특징을 적용하지 않은 기존 LFU 기법의 캐쉬를 사용했을 때가 8.9%의 성능 향상이 이루어 졌으며 SIP 의 특징을 적용한 UA 별 캐쉬를 사용했을 때는 캐쉬를 사용하지 않을 때 보다 12.25%, LFU 기법의 캐쉬를 사용했을 때보다 3.35%의 성능 향상이 있음을 확인 할 수 있다.

5. 결론 및 향후 연구 과제

본 논문에서 제안한 개별 캐쉬 기법은 UA 와 Proxy 서버 간의 동작 패턴을 적용하여 해당 UA 와 관련이 없는 목적 UA 의 정보 검색을 배제 하여 불필요한 정보 검색 시간을 줄임에 따라 단일 캐쉬에서의 목적 UA 의 관련 정보의 검색 시간과 비교하였을 때 배제된 정보 검색 시간만큼의 검색 시간 절약의 효과를 가져왔다. 즉 캐쉬를 사용하지 않을 때 보다 12.25%의 성능 향상 효과, LFU 기법의 캐쉬를 사용했을 때 보다 3.35%의 성능 향상 효과를 볼 수 있다. 이 결과에서 볼 때 목적 UA 의 정보 검색 시간을 줄임으로 해서 Proxy 서버의 call 메시지 처리 효율을 향상 시키는 결과를 가져 왔다.

제안된 개별 캐쉬 기법은 UA 별 캐쉬에 의해서 검색 시간이 일부 줄어 들어 성능을 향상 시키는 효과를 가져오지만 UA 별 캐쉬를 적용한 proxy 서버의 캐쉬 사이즈가 성능에 미치는 영향에 대한 연구와 registrar 에 도입한 LFU 캐쉬의 사이즈가 성능에 미치는 영향에 대한 연구가 필요하다. 또한 본 연구에서는 기존 SIP 시스템의 특징을 적용하지 않은 캐쉬 기법으로 LFU 캐쉬 기법과 제안 캐쉬 기법을 비교하여 테스트 하였다. 그러나 LFU 기법이 아닌 기존의 다른 기법을 적용한 캐쉬에 대한 비교 연구가 필요하다.

참고문헌

[1] M. Handley et al., “SIP: Session Initiation Protocol”, RFC2543, Mar. 1999. A
[2] Rosenberg, et. al. “SIP: Session Initiation Protocol“ , RFC3261, June 2002
[3] 이승훈, “메모리 참조 패턴 및 페이지 교체 기법”, 정보과학회논문지 VOL. 27 NO. 02 pp. 50 ~ 52 2000 . 10
[4] 반효경, “분산 이질형 객체 환경에서 캐슁 알고리즘의 설계 및 성능 분석”, 정보과학회논문지:시스템 및 이론 제 27 권 제 6 호, pp583~591, 2000.06
[5] 이종화, “SIP 기반 차세대 응용기술” 정보처리 제 8 권, pp 27 ~ 33 2001. 03
[6] 코리안 클릭, “제 6 차 인터넷 사용 실태 보고서” 2003.03
[7] 최종무, “적응력있는 블록 교체 기법을 위한 효율적인 버퍼 할당 정책”, 한국정보과학회논문지 A, VOL. 27 NO. 03 pp. 324 ~ 336 2000.03