중재 쓰레드와 지연 캐싱에 의한 스퀴드 프록시 서버 성능 향상에 관한 연구

이 대 성[†]·김 유 성[††]·김 기 창[†††]

요 약

인터넷 사용자의 폭발적인 증가로 인해 캐시 서버의 사용이 대두되고 있다. 이에 따라 캐시 서버의 성능을 향상시키기 위한 많은 연구들이 진행되었다. 본 논문에서는 캐시 서버의 병목 현상을 유발하는 원인을 분석하고 그에 대한 해결책으로 중재 쓰레드와 지연 캐싱을 제안한다. 중재 쓰레드를 사용함으로써 캐시 서버가 디스크 쓰기 연산 시에 준비된 다중 쓰레드를 탐색해야 하는 문제점을 없애고 즉시 사용자의 서비스 요청에 응답함으로써 보다 빠른 서비스를 제공할 수 있도록 한다. 또한 준비된 다중 쓰레드가 없는 경우에는 지연 캐싱을 이용해 과부하 상태의 디스크 연산을 피하고 큐가 안정화될 때 캐싱을 하도록 하여 시스템이 안정성 있게 동작하도록 하였다. 제안된 캐시 서버는 스퀴드(SQUID) 캐시 서버를 변형하여 구현되었고 실험을 통해 기존 스퀴드 캐시 서버의 성능과 비교한 결과, 응답 시간의 향상을 관찰할 수 있었다.

A Study on Improving SQUID Proxy Server Performance by Arbitral Thread and Delayed Caching

Dae Sung Lee[†] · Yoo-Sung Kim[††] · Ki Chang Kim[†††]

ABSTRACT

As the number of the Internet users increases explosively, a solution for this problem is web caching. So, many techniques on improving cache server performance have been suggested. In this paper, we analyze the cause of the bottleneck in cache servers, and propose an arbitral thread and delayed caching mechanism as a solution. We use an arbitral thread in order to provide a quick service to user requests through eliminating the ready multi-thread search problem in case of disk writing operation. we also use delayed caching in order to provide stable system operation through avoiding overloaded disk operation and queue threshold. Proposed cache server is implemented through modification on SQUID cache server, and we compare its performance with the original SQUID cache server.

키워드 : 캐시 서버(Cache Server), 캐시(Cache), 중재 쓰레드(Arbitral Thread), 지연 캐싱(Delayed Caching)

1. 서 론

미 국방성의 ARPANET으로부터 시작된 인터넷은 1990년대 이후로 WWW(World Wide Web)의 성장과 함께 그 사용자가 꾸준히 증가하고 있으며 특히 최근 몇년 동안 급격히 증가하는 추세이다. 이에 따른 네트웍 트래픽의 증가로 패킷 교환 시 일어나는 충돌에 의한 속도 저하와 서버 시스템의 오버헤드로 인한 속도 저하가 발생하게 되어 인터넷 서비스의 정체 현상이 가속화 되고 있다[1]. 더구나 기존의 텍스트 위주의 데이터에서 실시간 비디오, 오디오와 같은 대용량의 멀티미디어 데이터에 대한 사용량이 급증함에 따라 더 큰 네트웍 대역폭이 요구되고 있다. 그러나 네트웍 대역폭의 증가는 국가의 기간 산업과 연동되어 있기 때문에 경제적인 문제가 크게 부각이 된다. 실제로 네트웍 트래픽의 증가율에 비해 네트웍 대역폭의 증가율은 현저하게 미비하여 그 격차가 점점 벌어지고 있다[2]. 따라서 단순한 대역폭의 증가만으로는 인터넷 서비스의 정체 현상을 근본적으로 해결할 수 없다.

그러나 최근 사용자들의 서비스 요청은 일부 핫 오브젝트(hot object)에 편중되고 있으며 요청 페이지의 크기도 증가하고 있는 추세이다[3]. 더욱이 인터넷 방송국과 같은 실시간 멀티미디어 데이터를 처리해야 하는 새로운 방송 매체 서비스의 등장으로 대용량의 멀티미디어 데이터에 대한 서비스 요청이 급증하고 있다. 이러한 현상은 웹 캐시 기술(web cache technology)의 개발을 크게 부각시키고 있다. 웹 캐시 기술은 위에서 언급한 문제들을 해결하는 경제적이고

※ 이 논문은 한국과학재단 지역대학 우수과학자 지원 연구 지원에 의해 연구되었음(R05-2002-000-01048-0).
† 준 회 원 : 인하대학교 대학원 전자계산 공학과
†† 종신회원 : 인하대학교 정보통신 공학부 교수
††† 정 회 원 : 인하대학교 정보통신 공학부 교수
논문접수 : 2002년 8월 3일, 심사완료 : 2002년 11월 20일

효율적인 방법중의 하나이다. 웹 캐시 기술을 사용함으로써 사용자의 서비스 요청에 대한 응답 시간을 줄일 수 있고 실제 서버로의 요청을 지역 네트웍 안에서 해결함으로써 서버에 대한 부하 감소와 전체 네트웍 트래픽 또한 줄일 수가 있다[4].

이에 발맞추어 웹 캐시 서버의 개발이 활발히 진행되고 있으나 아직 많은 문제점들을 가지고 있다. 첫째, 일반적인 웹 캐시 서버들은 그 성능이 저조하여 캐시 서버 본연의 목적을 달성하지 못하고 있다. 캐시 서버는 지역 클라이언트들의 요청을 받아들이는 관문으로 그 성능이 저조할 경우 네트웍 서비스 속도의 향상보다는 오히려 지역 네트웍에 대한 또 다른 병목 현상을 유발하게 된다. 이러한 문제점을 해결하기 위해 캐시 서버 자체의 성능을 개선하고자 하는 연구가 진행되어 왔으며 그 예로 디스크 I/O 시간을 중첩 시킨 쓰레드를 사용하는 방법과 RAID(Redundant Arrays of Inexpensive Disks)를 이용하는 방법 등이 있다[5, 6]. 위의 연구들은 웹 캐시 서버의 특성(요청 데이터를 클라이언트에게 서비스하는 동안 웹 오브젝트를 디스크에 저장하는 일)을 간파하고 성능상의 문제점을 유발하는 디스크 I/O 연산을 효과적으로 수행하도록 제안된 연구들이다. 둘째, 캐시 서버는 일반 웹 서버와 달리 웹 오브젝트를 디스크에 저장해야 하기 때문에 서비스해야 할 클라이언트 수가 증가할 경우 저장 용량의 한계에 부딪히게 된다. 이와 같은 문제점을 해결하기 위해 많은 연구들이 수행되어 왔으며 그 대표적인 예로는 ICP(Internet Cache Protocol), Summary Cache, CARP(Cache Array Routing Protocol) 그리고 리다이렉션(redirection) 메시지를 이용하는 방법 등이 있다[1, 7-9]. 위의 연구들은 캐시 서버 자체의 성능 보다는 저장 용량의 한계를 극복하고 클라이언트의 요청을 지역적으로 분산하는 알고리즘을 수행하는데 비중을 두고 있다.

본 논문에서는 웹 캐시 서버 자체의 성능을 개선하는데 주력하였다. 이를 위해 웹 캐시 서버 벤치마킹 프로그램인 폴리그래프(Polygraph[14])를 통해 웹 캐시 서버의 성능을 저하시키는 요인을 분석하고 디스크 I/O 연산이 성능상의 문제점을 유발한다는 사실을 확인하였다. 이를 바탕으로 중재 쓰레드를 이용해 디스크 I/O 시간을 중첩시킴과 동시에 높은 요청율과 캐시 미스(Cache Miss) 시에도 효과적으로 대처할 수 있도록 지연된 캐싱(Delayed Caching) 방법을 적용하였다. 그리고 이 두 알고리즘을 구현한 시스템(Wise Cache)을 폴리그래프를 통해 실험하여 그 효율성을 검증하였다. 본 논문의 구성은 다음과 같다. 2장에서는 기존의 방법들을 소개하고 그 한계점을 지적한다. 3장에서는 공개 웹 캐시 서버인 스퀴드에서의 실험을 통해 디스크 I/O 연산이 병목 현상을 유발함을 실험을 통해 확인한다. 4장에서는 본 논문이 제시하는 중재 쓰레드를 이용해 디스크 I/O 시간을 효과적으로 중첩시키고 높은 요청율과 캐시 미스시에도 시스템이 안정성있게 운영되도록 하는 지연된 캐싱 방법을 소개한다. 5장에서는 제안된 시스템의 성능을 실험을 통해 분석하고 평가하며 6장에서 결론 및 향후 연구 방향에 대해 제시한다.

2. 관련 연구

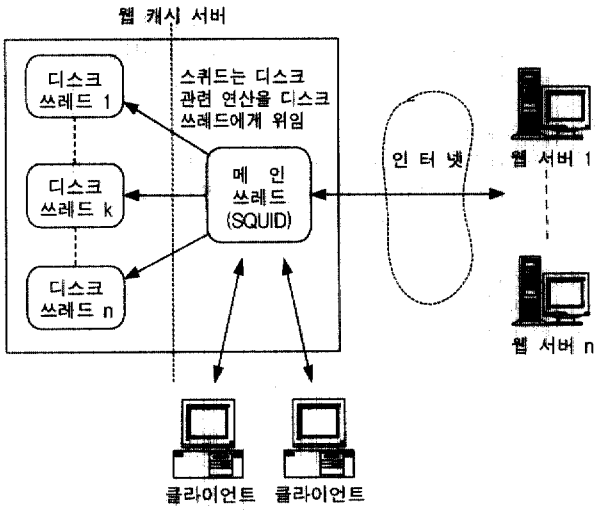
최근 인터넷 트래픽의 정체 현상이 가속 됨에 따라 웹 캐시 서버의 사용이 대두되고 있다. 그러나 웹 캐시 서버의 사용은 처리 요청의 집중에 따른 또 다른 병목 현상을 유발할 수 있기 때문에 웹 캐시 서버의 성능을 개선하고자 하는 연구들이 활발히 진행되고 있다. 그 예로 쓰레드를 이용해 디스크 I/O 시간을 중첩 시키는 방법과 RAID를 이용하는 방법 등이 있다[5, 6]. 이러한 연구들과 더불어 보스턴 대학의 인터넷 캐시에 관한 연구에 의하면 적정 히트율(hit ratio)을 유지하기 위해서는 캐시 서버 사용자의 증가에 따라 캐시의 크기 또한 증가해야 한다고 보고하고 있다[10]. 이러한 용량 문제의 해결과 각 사용자의 요청을 분산시킴으로써 캐시 서버의 성능을 개선하고자 하는 연구들이 진행되어 왔다. 그 대표적인 예로 ICP와 Summary Cache, CARP, 리다이렉션 메시지를 이용하는 방법 등이 있다[1, 7-9]. 본 절에서는 본 논문과 관련이 있는 디스크 I/O 최적화에 관한 연구들을 중심으로 살펴보겠다.

2.1 쓰레드를 이용해 디스크 I/O 시간을 중첩시키는 방법

다중 쓰레드를 사용해 디스크 I/O 시간을 중첩시킴으로써 응용 프로그램의 디스크 연산에 대한 오버헤드를 줄이고자 하는 연구가 진행되었다[5]. 이와 같은 연구는 디스크 I/O 연산을 효과적으로 수행해야 하는 웹 캐시 서버의 경우에 적절히 적용될 수 있다. 실제로 공개 웹 캐시 서버인 스퀴드 프로그램의 경우 다중 쓰레드를 이용해 디스크 연산을 수행함으로써 성능을 개선시키고 있다. 다중 쓰레드를 사용한 스퀴드의 동작 원리는 (그림 1)과 같다.

(그림 1)에서 알 수 있듯이 쓰레드를 이용한 스퀴드는 캐시 미스시에 웹 서버로부터 받은 응답 패킷을 디스크에 저장하는데 있어서 자신의 자식 쓰레드에게 디스크 쓰기 연산을 위임 시키고 있다. 또한 캐시 히트 시에도 자식 쓰레드를 이용해 캐시된 데이터를 읽어 온후 스퀴드가 클라이언트에게 서비스하도록 하고 있다. 이러한 접근 방법은 웹 캐시 서버의 병목 현상을 일으키는 디스크 연산을 자식 쓰레드에게 위임함으로써 성능을 개선하려 하고 있다. 그러나 클라이언트의 서비스 요청이 폭주하여 자식 쓰레드들이 모두 디스크 관련 연산을 수행하게 되면 그 이후의 스퀴드 동작은 디스크 연산에 관여하지 않는 준비된 쓰레드가 나타날 때까지 탐색을 해야 한다. 이는 높은 서비스 요청에 대해 효과적으

로 대처하지 못하는 문제점을 나타낸다. 실제로 초당 350개의 요청에 대해서는 준비된 쓰레드를 탐색하는 시간 지연 때문에 요청 서비스에 대한 응답 시간이 지연됨을 캐시 서버 벤치마킹 프로그램인 폴리그래프를 통해 확인할 수 있었다.

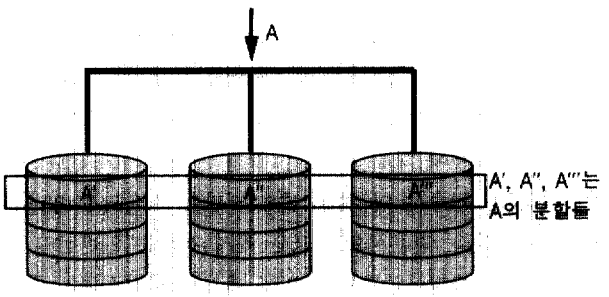


(그림 1) 디스크 쓰레드를 이용한 I/O 시간 중첩

2.2 RAID를 이용하는 방법

RAID(Redundant Arrays of Inexpensive Disks)는 저 가격을 유지하면서 입출력 성능을 향상시키기 위한 여러 해결책 중에서 저장 장치의 속도와 신뢰성을 고려한 것이다[6]. RAID 시스템은 여러 드라이브의 집합을 하나의 저장 장치로 생각하고 장애가 발생했을 때 저장된 데이터를 잃어버리지 않도록 하며 각각의 드라이브가 독립적으로 동작할 수 있도록 하고 있다. RAID를 사용함으로써 데이터의 가용성을 높이고 데이터를 보호할 수가 있으며 저렴한 비용으로 대용량의 디스크를 구현할 수 있다. 또한 하나의 디스크 입출력 요구에 대하여 여러 디스크에 데이터를 분산시키고 병렬적으로 입출력을 처리함으로써 시스템의 효율성을 증가시킬 수 있는 장점이 있다. RAID는 데이터와 패리티 정보를 디스크에 배치하는 방법에 따라 RAID 레벨(0~5 레벨)을 구분하고 있다. RAID를 웹 캐시 서버의 성능 향상을 위해 사용할 경우, 디스크 연산 속도를 높이기 위해서는 RAID 레벨 0(스트라이핑 기능)를 사용해야 한다. RAID 레벨 0는 (그림 2)에서 보는 바와 같이 하나의 데이터를 동시에 나누어서 기록 하기 때문에 속도가 빠르다. 그러나 디스크가 많을수록 성능은 올라가지만 그 만큼의 안정성이 떨어지게 된다. 즉, 디스크의 수가 증가할수록 그 중 하나의 디스크가 장애를 일으킬 확률이 증가하는 것이다.

따라서 웹 캐시 서버의 디스크로써 RAID를 사용할 경우 캐시된 데이터가 손상되지 않도록 디스크 장애에 대한 대비가 필요하다. 이와 더불어 RAID 장비를 효과적으로 이용하기 위해서는 고가의 비용이 드는 문제점이 있다.



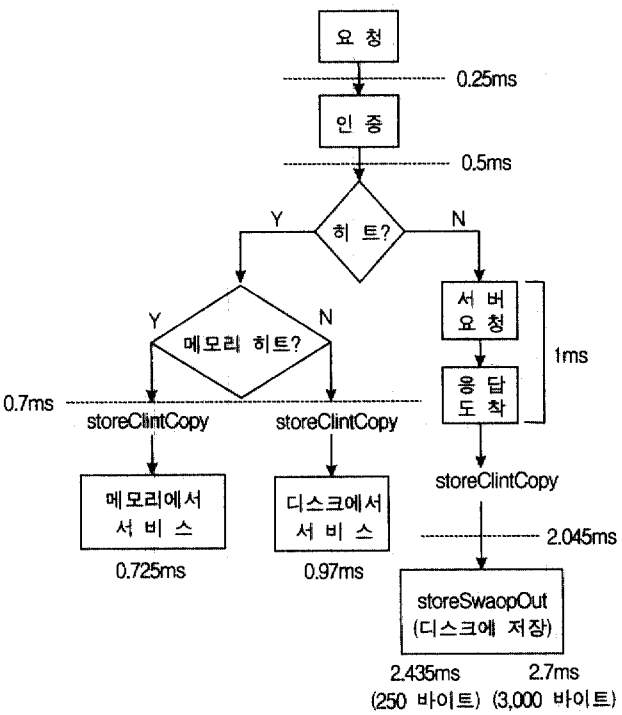
(그림 2) RAID 레벨 0

3. 스퀴드 성능 분석

스퀴드는 Harvest 프로젝트[7]의 일환으로 개발된 소프트웨어로서 현재 전세계적으로 널리 사용되고 있는 캐시 서버이다. 본 절에서는 스퀴드의 성능을 주요 부분별로 분석함으로써 스퀴드에서 병목 현상을 유발하는 부분과 그 문제점을 알아본다.

3.1 각 부분별 스퀴드 실행 소요 시간 측정

본 항목에서는 스퀴드의 각 부분별 실행 소요 시간을 측정한다. 이를 위해 본인이 직접 제작한 URL 생성기를 이용하여 2,000개의 서비스 요청을 스퀴드에 송신하고 그 평균 응답 시간을 측정한다. 각 부분별 응답 시간의 측정은 스퀴드 코드의 주요 부분에서 클라이언트에게 응답 패킷을 송



(그림 3) 각 부분별 스퀴드 실행 소요 시간 측정

신하도록 한 후에 최종적으로 클라이언트에서 2000개의 요청에 대한 평균 응답 시간을 측정하였다.

(그림 3)은 각 부분별 응답 시간을 측정한 결과이며 store SwapOut 루틴에서 디스크에 오브젝트를 저장하는 시간이 상대적으로 많이 소요되고 있음을 알 수 있다. 또한 3,000바이트의 응답 패킷에 대해 더 많은 실행 시간이 요구된다. 즉, 파일 크기가 클수록 더 많은 실행 시간이 소요되고 있음을 확인할 수 있다. 결론적으로 모든 캐시 서버는 일반적인 웹 서버와 달리 클라이언트에게 요청 패킷을 서비스하는 동안 웹 오브젝트를 디스크에 저장해야 하는 추가적인 임무 수행 때문에 병목 현상이 발생하고 있음을 예측할 수 있다.

4. WiseCache : 중재 쓰레드와 지연 캐싱을 이용한 캐시 서버

3장에서 각 부분별 시간 측정을 통해 알수 있듯이 웹 캐시 서버는 일반 웹 서버와는 달리 웹 오브젝트를 디스크에 저장해야 하는 문제점이 있으며 이 때문에 성능상에 문제가 있음을 확인하였다. 본 절에서는 중재 쓰레드와 지연 캐싱을 이용해 그러한 문제점에 대한 해결책을 제시한다. 중재 쓰레드는 요청이 폭발적으로 증가할 경우 생겨나는 다중 쓰레드의 탐색 문제를 해결하고 보다 빠른 클라이언트에의 송신을 위해 제시되었다. 이를 위해 디스크 I/O 데이터를 관리하기 위한 큐(queue)를 생성하고 메인 쓰레드는 큐의 생산자로써 디스크 I/O 관련 메타(meta) 정보를 큐에 삽입하며 중재 쓰레드는 큐의 소비자로써 삽입된 정보를 큐에서 삭제하여 다중 쓰레드(실제 디스크 I/O를 수행하는 쓰레드)에게 디스크 I/O 연산을 위임하도록 하였다(그림 5). 지연 캐싱은 클라이언트의 높은 요청율로 인해 준비된 다중 쓰레드가 소멸되고 큐의 길이가 한계 값을 초과하는 시스템 과부하 상태를 초래하지 않도록 하여 시스템이 안정적으로 동작하도록 하였다. 즉, 클라이언트의 요청이 폭주하여 큐의 길이가 한계 값을 초과할 경우 요청 데이터를 캐시하지 않고 단지 요청 데이터에 대한 메타 정보만을 기록하고 일정 시간이 경과한 후 시스템이 안정된 상태로 돌아왔을 때 기록된 메타 정보를 보고 웹 서버에게 재 요청을 하는 것이다. 이는 요청 데이터에 대한 캐시 포기로 인해 특정 사용자는 캐시 미스에 의한 응답 시간의 지연이 발생하지만 다수의 사용자는 시스템의 안정화에 따른 서비스를 받을 수 있게 된다.

중재 쓰레드 구현을 위한 기본 알고리즘은 다음과 같다.

```
입력 : NTHREAD                          /* 다중 쓰레드의 개수 */
출력 : 다중 쓰레드, DiskQ, 중재 쓰레드
for (i = 0 ; i < NTHREAD ; i++)
   디스크 I/O 쓰레드(다중 쓰레드) 생성 ;
DiskQ = (qcond_t * )malloc(sizeof(qcond_t)) ;
→ 큐 관리를 위한 자료 구조를 생성한다.
DiskQ → mutex = &mutex ;            /* 쓰레드간의 상호 배제를
                                       보장하는 mutex */
DiskQ → condition = &condition ;    /* 쓰레드간의 Condition
                                       Variable */
pthread_create(&Arg_Thread, NULL, arbitral_Th, NULL) ;
→ 중재 쓰레드를 생성한다.
...
if (Cache Hit) {
   storeReadData 구조체를 생성하고 디스크 읽기 연산에 필요한
   메타 정보를 생성된 storeReadData 구조체에 담는다. ;
   Q_push(DiskQ, storeReadData) ;
}
else { // Cache Miss
    storeWriteData 구조체를 생성하고 디스크 쓰기 연산에 필요한
    메타 정보를 생성된 storeWriteData 구조체에 담는다. ;
    Q_push(DiskQ, storeWriteData) ;
}
```

(알고리즘 1) 메인 쓰레드

(알고리즘 1)은 WiseCache의 메인 쓰레드의 역할을 나타낸 것으로 큐 관리자를 초기화하고 중재 쓰레드를 생성한다. 또한 캐시 히트나 미스 시에 중재 쓰레드가 사용할 메타 정보들을 큐에 삽입한다. 이와 관련된 주요 자료 구조들은 다음과 같다.

```
struct qcond_t {                    /* 디스크 I/O 연산시 큐 관리를
                                       위한 구조체 */
pthread_mutex_t * mutex ;           /* Race Condition 방지를 위한
                                       구조체 포인터 */
pthread_cond_t * condition ;        /* Condition Variable 구조체
                                       포인터 */
qmember_t * queueHead ;             /* qmember_t 구조체 포인터 */
qmember_t * queueTail ;             /* qmember_t 구조체 포인터 */
} ;                                 /* end of qcond_t */
```

(자료 구조 1) qcond_t

```
struct qmember_t {                  /* 디스크 I/O를 위한 큐 멤버 */
void * data ;                       /* 메타 정보(storeWriteData, storeReadData)
                                       구조체에 대한 포인터 */
qmember_t * next ;                  /* qmember_t 구조체에 대한 포인터 */
} ;
```

(자료 구조 2) qmember_t

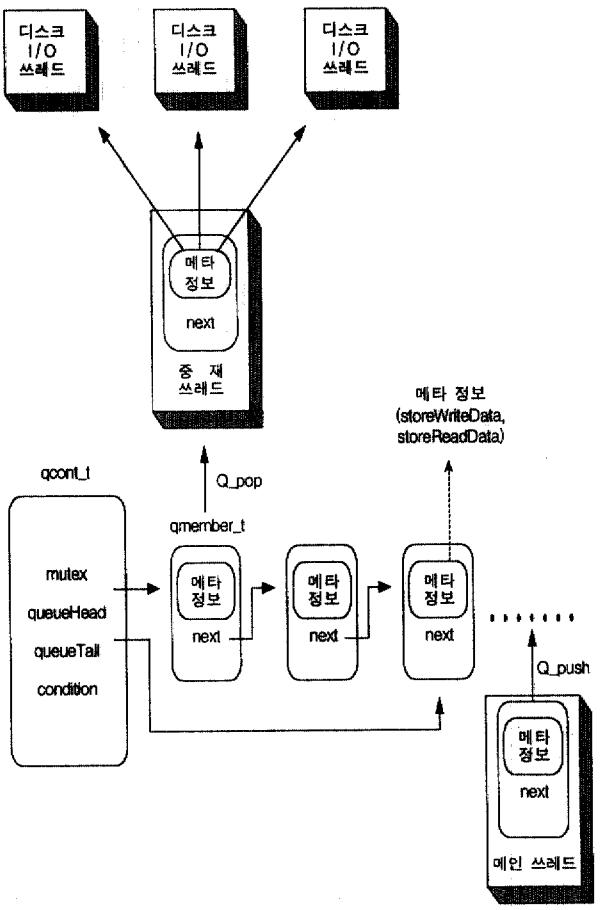
```
struct _storeWriteData {            /* Disk Write에 대한 메타 정보
                                       구조체 */
int o_flag ;                        /* 연산(읽기 : 0, 쓰기 : 1) 플래그 */
StoreEntry * entry ;                /* 캐시 데이터의 메타 정보
                                       구조체에 대한 포인터 */
store_status_t store_status ;       /* 캐시 상태에 대한 정보 */
off_t swap_offset ;                 /* 현재 캐시된 데이터의 offset
                                       정보 */
```

```
storeIOState * sio;          /* 캐시 파일의 상세 정보
                                구조체에 대한 포인터 */
char * buf                   /* 캐시할 데이터에 대한 포인터 */
size_t size;                 /* 캐시할 데이터의 크기 */
off_t offset;                /* 캐시할 데이터의 offset 정보 */
Free * free_func;            /* 캐시후 메모리를 free할 함수에
                                대한 포인터 */
}
```

(자료 구조 3) _storeWriteData

```
입력 : DiskQ, 메타 정보(큐에 삽입된 storeWriteData 또는
       storeReadData)
출력 : 없음
static void * arbitral_Th(void * arg) {
    void * store_data = NULL;
    while ((store_data = Q_pop(DiskQ)) != NULL){
        if (store_data == storeWriteData)
            디스크 쓰기 연산을 다중 쓰레드 (Disk I/O Threads)
            에게 위임
        else   /* store_data == storeReadData */
            디스크 읽기 연산을 다중 쓰레드 (Disk I/O Threads)
            에게 위임
        free((void * ) store_data);
        store_data = NULL;
    }
}
```

(알고리즘 2) 중재 쓰레드(Arbitral Thread)



(그림 5) 중재 쓰레드를 이용한 WiseCache 동작 유형

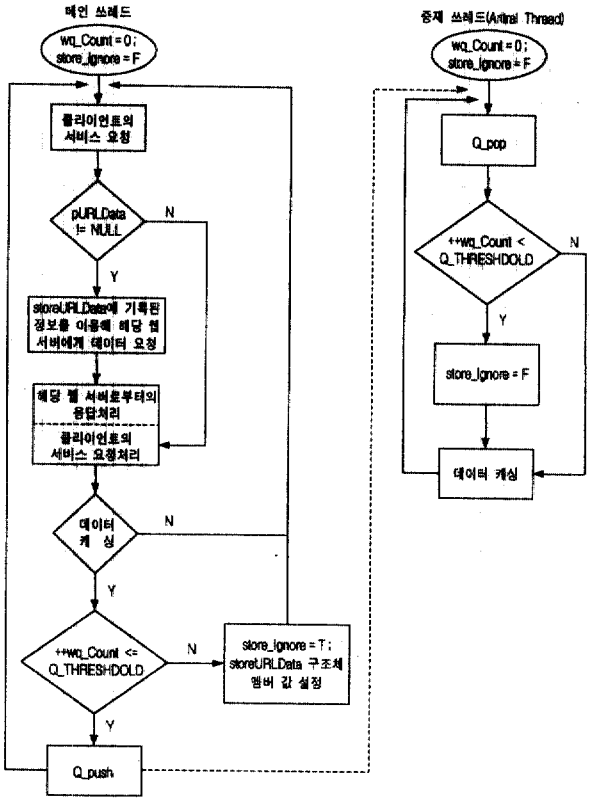
(알고리즘 2)는 중재 쓰레드의 구현을 나타낸다. 중재 쓰레드는 큐에 데이터가 삽입되면 이를 꺼내어 다중 쓰레드에게 디스크 I/O 연산을 위임하고 큐에 삽입된 데이터가 없을 경우에는 Q_pop(DiskQ) 루틴 내에서 데이터가 삽입될 때까지 기다린다. (그림 5)는 중재 쓰레드를 이용한 WiseCache의 전체적인 동작 유형을 구체적으로 나타낸 것이다. 중재 쓰레드와 메인 쓰레드는 위에서 정의한 자료 구조들을 이용하여 서로 통신을 하게 된다. (그림 5)에서 알 수 있듯이 메인 쓰레드는 디스크 I/O 연산이 필요한 경우에 단지 디스크 I/O에 필요한 메타 정보만을 큐에 삽입하고 계속 진행하게 되며 중재 쓰레드는 삽입된 디스크 I/O 메타 정보를 큐에서 꺼내어 다중 쓰레드에게 실제 디스크 I/O 연산을 위임하게 된다. 따라서 클라이언트의 요청이 폭주하여 준비된 다중 쓰레드(디스크 I/O 쓰레드)가 없다 하더라도 중재 쓰레드가 다중 쓰레드를 폴링하면서 기다리게 되며 이에 따라 메인 쓰레드는 디스크 I/O 연산과 상관없이 자신의 임무를 계속 수행할 수 있게 된다.

지연 캐싱 구현을 위한 기본 알고리즘은 다음과 같다.

```
extern Boolean store_Ignore = FALSE;
extern int wq_Count = 0;
extern int Q_THRESHOLD = 다중 쓰레드 수(NTHREAD)
   / 캐시 디스크 수 / 2;
/* 다중 쓰레드 수 / 캐시 디스크 수 == 디스크당 쓰레드 수 */
/* 디스크당 쓰레드 수 / 2 == 디스크 읽기, 쓰기당 쓰레드 수 */
...
if (storeURLData 구조체가 존재 && store_Ignore == FALSE)
    storeURLData 구조체에 기록된 URL을 이용해 해당
    웹 서버에게 재 요청
if (Cache Miss){
  if (++wq_Count <= Q_THRESHOLD && store_Ignore ==
     FALSE)
     Q_push(DiskQ, storeWriteData);
  else {
     wq_Count--;
     store_Ignore = TRUE;
     storeURLData 구조체를 생성하고 웹 서버에게 재 요청할 메타
     정보를 storeURLData 구조체에 기록
  }
}
```

(알고리즘 3) 지연 캐싱(메인 쓰레드)

(알고리즘 3)은 큐에 삽입된 멤버의 개수가 한계 값(Q_THRESHOLD)을 초과할 경우에 Q_push(DiskQ, storeWriteData)를 하지않고 즉, 캐싱을 포기하고 이 데이터에 대한 메타 정보를 storeURLData에 기록하게 되며 시스템이 여유가 있을 때(store_Ignore = FALSE의 경우) storeURLData에 기록된 정보를 이용하여 해당 웹 서버에게 재 요청을 하도록 구현되었다. (그림 6)은 중재 쓰레드를 이용해 지연 캐싱을 구현한 전체적인 내용을 보여준다.



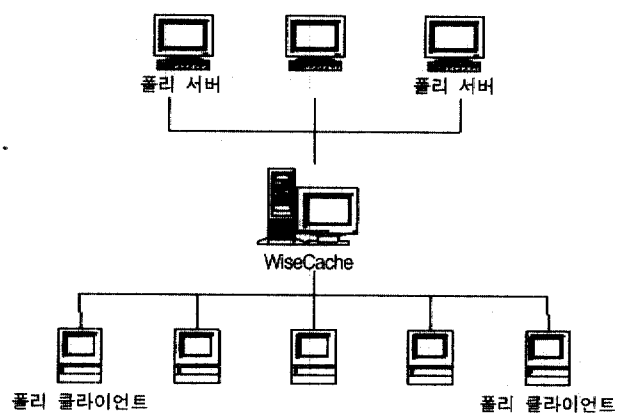
(그림 6) 지연 캐싱

5. 실험 및 성능 평가

본 절에서는 4장에서 설명한 중재 쓰레드와 지연 캐싱을 구현한 WiseCache의 성능 향상을 검증한다. WiseCache의 구현은 스퀴드 코드의 변경을 통해 이루어 졌으며 폴리그래프를 이용해 WiseCache의 성능을 측정하였다.

5.1 실험 시스템의 구성

본 실험에서 WiseCache는 squid-2.3.STABLE3 버전을 변경하여 구현하였고 RedHat LINUX 6.2의 운영 체제를 사용하였다. WiseCache의 하드웨어 사양은 CPU 700MHz, Memory 768MB, HDD(EIDE) 5400rpm 20GB×4의 PC가 사용되었다. 또한 폴리그래프는 2.5.4-STABLE 버전을 사용하였으며 모든 실험은 100Mbps에서 실행하였다.

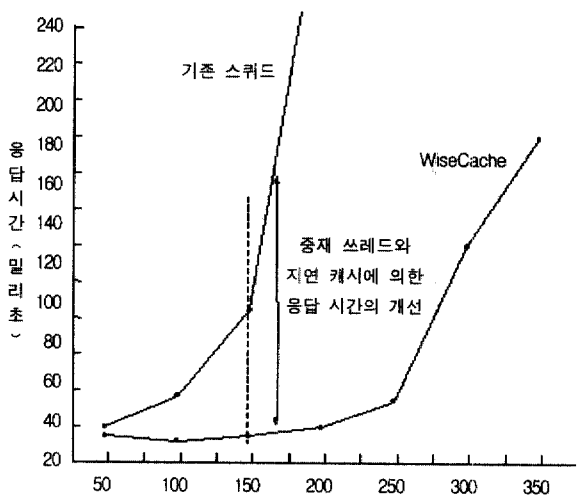


(그림 7) 실험 환경

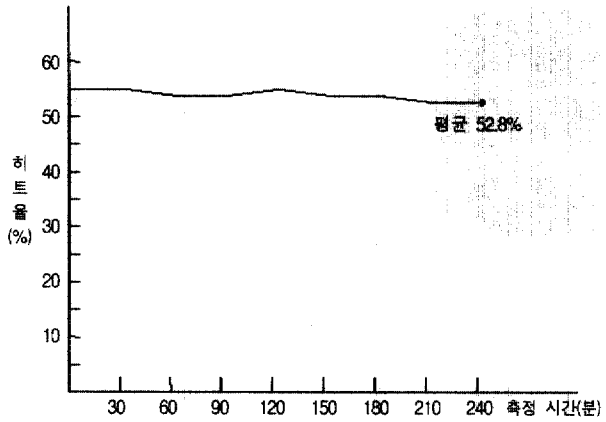
(그림 7)은 실험을 위한 시스템의 구성을 나타낸다. 본 실험에서 WiseCache는 250Mbytes의 캐시 메모리를 사용하였고 다중 쓰레드는 64개를 이용하였다.

5.2 실 험

본 실험에서는 4장에서 구현한 중재 쓰레드와 지연 캐싱의 효율성을 검증한다. 이를 위해 기존의 스퀴드 성능과 WiseCache의 성능을 폴리그래프를 이용해 비교 분석한다. 본 실험은 (그림 7)의 실험 환경에서 4시간 동안 실행되었고 요청율에 대한 평균 응답 시간과 히트율을 측정하였다. 폴리그래프의 트래픽 모델은 Constant-Rate 모델[14]을 적용하였으며 히트 율은 55%, 응답 패킷의 크기는 6킬로바이트로 고정시켰다. 다음의 (그림 8)은 폴리그래프를 이용해 기존 스퀴드의 성능과 WiseCache의 성능을 테스트한 결과이며 (그림 9)는 초당 200개의 요청에 대한 4시간 동안의 히트율 변화를 나타낸다. (그림 8)에서 WiseCache는 클라이언트의 요청이 폭주하더라도(150 요청/초 이상) 응답 시간을 효과적으로 개선시키고 있다. 이는 중재 쓰레드와 지연 캐싱을 통해 캐시 서버의 성능이 향상되었음을 보여준다. (그림 9)는 지연 캐싱을 이용한 WiseCache의 4시간 동안의 히트율을 측정한 것으로 지연 캐싱에 의해 다소 히트율이 떨어지는 현상(약 2.2% 정도)을 관찰할 수 있다. 그러나, 지연 캐싱에 의해 (그림 8)에 나타난 바와 같이 성능이 향상됨을 알 수 있다. 이는 클라이언트의 요청이 폭주하여 준비된 다중 쓰레드가 소진되고 큐의 길이가 한계 값을 초과할 경우 캐싱을 포기하고 시스템이 안정화 되었을 때 캐싱을 하도록 한 것이다.



(그림 8) 기존 스퀴드와 WiseCache의 응답 시간 비교



(그림 9) WiseCache의 히트율 변화

6. 결론 및 향후 연구 방향

본 논문에서는 캐시 서버 자체의 성능을 개선하는 데 주력하였으며 이를 위해 기존 캐시 서버를 대상으로 그 문제점과 원인을 찾아내고 그에 대한 해결책을 강구하였다. 본 논문에서 제안하는 WiseCache는 기존 캐시 서버의 문제점을 해결하기 위해 중재 쓰레드와 지연 캐싱을 이용하였다. 중재 쓰레드를 사용함으로써 캐시 서버가 디스크 연산 시에 준비된 다중 쓰레드를 탐색해야 하는 문제점을 없애고 즉시 사용자의 서비스 요청에 응답함으로써 보다 빠른 서비스를 제공할 수 있도록 한다. 또한 지연 캐싱을 이용해 과부하 상태의 디스크 연산을 피하고 큐가 안정화될 때 캐싱을 하도록 하여 시스템이 안정성 있게 동작하도록 하였다.

향후 연구 과제로는 사용자의 서비스 요청에 대해 보다 빠른 응답 시간을 제공하도록 캐시 서버의 히트 율을 높여야 하며 일반적인 서버들에서 나타나는 네트웍 I/O 대기 시간을 개선할 수 있는 연구들이 필요할 것이다.

참 고 문 헌

[1] Radhika Malpani, Jacob Lorch and David, "Making World Wide Web Caching Servers Cooperate," Berger University of California at Berkeley, http://bmrc.berkeley.edu/papers/1995/138/paper-59.html, 1995.

[2] Jeffrey K. MacKie-Mason and Hal R. Varian, "Some economics of the Internet," In 10th Michigan Public Utility Conference at Western Michigan University, November, 1992.

[3] Martin F. Zrlitt and Carey L. Williamson, "Internet Web Servers : Workload Characterization and Performance Implications," IEEE Transactions of on Networking, Vol.5, No.5, 1997.

[4] Van Jacobson, "How to kill the Internet," in SIGCOM '95 Middleware Workshop, ftp://ftp.ee.lhl.gov/talks/vj-web-flame.ps.Z, August, 1995.

[5] Sachin More and Alok Choudhary, "MTIO, A MULTI-THREADED PARALLEL I/O SYSTEM," in Proceeding of 11th International Parallel Proceeding Symposium, pp.368-373, April, 1997.

[6] David A. Patterson, Garth Gibson and Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks(RAID)," in SIGMOD Conference, pp.109-116, 1988.

[7] Duane Wessels and k. claffy, "ICP and the Squid Web Cache," http://ircache.nlanr.net/Cache/ICP, August, 1997.

[8] Li Fan, Pei Cao, Jussara Almeida and Andrei Z. Broder, "Summary Cache : A Scalable Wide-Area Web Caching Sharing Protocol," ACM Computer Communication Review, Vol.24, No.4, 1998.

[9] Microsoft Corporation, CARP White Paper, http://www.microsoft.com/proxy/beta/moreinfo.html, 1997.

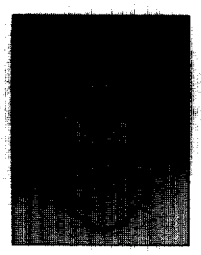
[10] Azer Bestavros, Robert L. Carter, Carlos R. Cunba Mark E. Crovella, Abdelsalam Heddaya and Sulaiman A. Mirdad, "Application-level document caching in the Internet," In Proceeding of the Second IEEE International Workshop on Services in Distributed and Networked Environments, pp.166-173, June, 1995.

[11] Carlos R. Cunba, Azer Bestavros and Mark E, Crovella, "Characteristaics of WWW client-based traces," Technical report, BU-CS-96-010, Boston University, October, 1995.

[12] Pietsch, Caching in the Washington state k-20 network, In the 2nd Web Caching Workshop, Boulder, Colorado, June, 1997.

[13] Kirby Beck, Tennessee cache box project, In the 2nd Web Caching Workshop, Boulder, Colorado, http://ircache.nlanr.net/Cache/Workshop97/, June, 1997.

[14] Alex Rousskov and Duane Wessels, "High Performance Benchmarking With Web Polygraph," http://polygraph.ircache.net/doc/papers/paper01.ps.gz, June, 2000.



이 대 성

e-mail : blue@super.inha.ac.kr

1999년 인하대학교 전자계산 공학과 (학사)

2001년 인하대학교 대학원 전자계산 공학과 (공학석사)

2000년~2001년 (주)softzone 기술 연구소 주임 연구원

2001년~2002년 (주)SOVIK 기술 연구소 선임 연구원

2001년~현재 인하대학교 대학원 전자계산 공학과(박사 과정)

관심분야 : 무선 LAN 보안, 실시간 운영체제, IPV6 등

김 유 성

e-mail : yskim@inha.ac.kr

1986년 인하대학교 전자계산 공학과 (이학사)

1988년 한국과학기술원 전산학과 (공학석사)

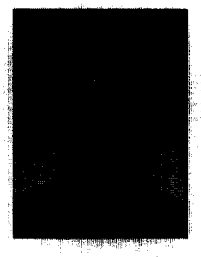
1992년 한국과학기술원 전산학과 (공학박사)

1990년~19992년 삼성전자 컴퓨터부문 주임 연구원

1996년~1997년 미국, 퍼듀대학교 전산학과 방문 연구원

1992년~현재 인하대학교 정보통신 공학부 부교수

관심분야 : 멀티미디어 정보검색, XML, 이동 데이터베이스



김 기 창

e-mail : kchang@inha.ac.kr

1986년 California State Politech대 전산학 (학사)

1988년 University of California, Irvine 전산학(석사)

1992년 University of California, Irvine 전산학(박사)

1992년~1994년 IBM T. J. Watson 연구소 연구원

1994년~1999년 인하대학교 전자계산 공학과 조교수

2000년~2002년 인하대학교 전자계산 공학과 부교수

2002년~현재 인하대학교 정보통신 공학부 부교수

관심분야 : 인터넷 보안, 운영체제, Wearable Computing 등