論文 97-22-2-1

분산 디렉토리 환경 하에서 효율적인 캐시 메카니즘 설계

正會員 李 康 雨*, 李 載 昊*, 林 海 喆**

Design of Cache Mechanism in Distributed Directory Environment

Kang Woo Lee*, Jae Ho Lee*, Hae Chull Lim** *Regular Members*

요 약

본 논문에서는 분산 디렉토리 환경에서 질의 처리 속도를 향상 시키기 위하여 원격지의 객체에 대한 질의와 결과를 요청지의 캐시에 저장하는 캐싱 메카니즘을 설계하였으며, 설계 단계는 다음과 같이 6단계로 나누어 진행하였다. 첫째, 분산 디렉토리 시스템에 저장되는 캐시 정보를 응용 데이타 정보, 시스템 데이타 정보, 메타 데이타 정보로 분류하였다. 둘째, 분류된 캐시 정보를 기반으로 캐시 시스템 구조를 설계하였다. 셋째, 각각의 캐시 정보에 대한 저장 구조를 설계하였다. 넷째, 데이타 캐시(응용 데이타 캐시, 시스템 데이타 캐시)의 대체 알고리즘으로 거리 정보와 접근 회수를 가중치로 부여한 최소-TTL 대체 알고리즘을 제안하였다. 다섯째, 질의에 대한 탐색 공간의 범위를 좁힘으로써 질의 처리 속도를 향상시키기 위해 이전 질의를 재구성한 메타 데이타 트리를 저장하는 메타 데이타 캐시의 운영 알고리즘을 개발하였다. 마지막으로, 제안된 캐시 메카니즘과 타 메카니즘과의 성능 평가를 수행하여 제안된 메카니즘의 우수성을 입증하였다.

ABSTRACT

In this paper, we suggest a cache mechanism to improve the speed of query processing in distributed directory environment. For this, request and result about objects in remote site are stored in the cache of local site. A cache mechanism developed through six phases; 1) Cached information which stored in distributed directory system is classified as application data, system data and meta data. 2) Cache system architecture is designed according to classified information. 3) Cache schema are designed for each cache information. 4) Least-TTL algorithms which use the weighted value of geograpical information and access frequency for replacements are developed for data caches(application cache, system cache). 5) Operational algorithms are developed for meta data cache which has meta data tree. This tree is based on the information of past queries and improves the speed of query processing by

*서남대학교 전산정보학과
**인천교육대학교 컴퓨터교육과
***홍익대학교 컴퓨터공학과
論文番號:97026-0122
接受日字:1997年 1月 22日

reducing the scope of search space. 6) Finally, performance evaluations are performed by comparing with proposed cache mechanism and other mechanisms.

I. 서 론

현대 통신망에서는 엄청난 수의 노드가 전세계적으로 분산되어 있으며, 이러한 노드를 구성하는 하드웨어와 소프트웨어는 이질적인 성격을 가지고 복잡하게 구성된다. 반면, 통신망이 복잡해지고 커지면서 관심 대상에 대한 정보의 양은 기하급수적으로 증가하고 있어, 대량의 분산데이타를 효과적으로 관리할 수 있는 분산정보 저장소의 필요성이 대두되고 있다. 이와 같은 요구 사항에 부응할 수 있는 강력한 후보 기술 중 하나가 ITU(International Telecommunication Union)의 X.500 디렉토리 시스템이다[1][2].

X.500 디렉토리 시스템은 객체(사람, 응용, 장비 등)에 관한 정보를 관리하며, 그러한 정보에 접근하는 응용 프로그램을 위한 구조적 메카니즘을 제공하고 있다. 특히, 이러한 서비스는 대규모 분산 컴퓨팅 환경에서는 중요한 기능이라 할 수 있다. 디렉토리의 정보는 디렉토리 정보 베이스(Directory Information Base; DIB)에 저장되며, DIB는 각각의 객체에 대한 정보를 포함하는 엔트리들로 구성된다. 디렉토리의 엔트리는 디렉토리 정보 트리(Directory Information Tree; DIT)라는 계층적 이름 공간(hierarchical name space)으로 표현된다[1].

전세계적으로 분산되어 있는 분산 디렉토리에서 객체(전자우편, MHS, FTAM 등)를 검색하는 명명 서비스(naming service)를 위해서는 통신망의 전체적인 구성을 파악하여야 하며, 객체에 대한 유일한 경로 명을 추측해야 하거나 모든 경로를 항해하여야 한다. 그러나 사용자는 모든 관심 대상이 되는 객체에 대한 통신망의 구성이나 경로 명을 알기가 어렵고, 또한 사용자가 객체에 대한 경로 명을 모른다면 수많은 데이타 저장소를 탐색해야 하는 어려움이 발생한다. 이와 같은 문제를 극복하기 위한 대안으로 서술 명명 서비스(descriptive naming service)라는 비절차적 질의 언어를 사용하는 방법이 연구되고 있다[4]. 서술 명명 서비스는 예를 들어, "select * from people where name ='kwlee'"와 같이 객체의 애트리뷰트를 기술하여 객체를 검색하는 방법으로, 사용자는 복잡하게 뒤얽힌 정보의 이름 공간 구조를 이해할 필요가 없이 객체에 접근할 수 있게 하는 방법이다. 그러나 서술 명명 서비스는 성능 면에서 낮은 단점을 가지고 있다. 일반적으로 사용하는 X.500의 탐색 기능은 서술 명명 서비스로서 전역 선택(global selection) 질의라 할 수 있다. 전역 선택 질의에 대한 결과를 얻기 위해서는 수백, 수천의 데이타베이스를 탐색해야만 한다[3][1]. 이러한 질의 성능 저하에 대한 해결책으로는 원격지에 있는 객체의 주소 정보를 요청지의 캐시에 저장하는 이름 캐싱(name caching) 방법이 있는데, 이 방법을 이용하면 지역 이름에 대한 이름 검색 비용을 줄임으로써 질의의 성능 향상을 기대 할 수 있다. 그러나 현재까지 이 분야에 대한 관련 연구와 X.5xx 시리즈 권고 안에서는 중복에 대한 연구만 이루어지고 있을 뿐 캐싱에 관한 연구가 이루어지고 있지 않는 실정이다.

따라서, 본 논문에서는 분산 디렉토리 환경에서 질의 속도를 높이기 위해 원격지의 객체에 대한 질의와 결과를 요청지의 캐시에 저장하는 캐싱 메카니즘을 설계하였다. 분산 디렉토리 시스템에 저장되는 캐시 정보를 응용 데이타 정보, 시스템 데이타 정보, 메타 데이타 정보로 분류하고, 분류된 캐시 정보를 기반으로 캐시 시스템 구조와 각각의 캐시 정보에 대한 저장 구조를 설계하였다. 또한, 데이타 캐시(응용 데이타 캐시, 시스템 데이타 캐시)의 대체 알고리즘으로 거리 정보와 접근 횟수를 가중치로 부여한 최소-TTL 대체 알고리즘을 제안하였다. 질의에 대한 탐색 공간의 범위를 좁힘으로써 질의 속도를 향상시키기 위해 이전 질의를 재구성한 메타 데이타 트리를 저장하는 메타 데이타 캐시의 운영 알고리즘을 개발하였다. 마지막으로, 제안된 캐시 메카니즘과 타 메카니즘과의 성능 평가를 수행하여 제안된 메카니즘의 우수성을 입증하였다.

본 논문의 구성은 다음과 같다. 제 2장은 X.500에서 제안한 디렉토리 정보의 분산 방법과 Sprite 시스템에서의 객체 탐색 그리고 분산 시스템의 성능 향상을 위한 캐싱을 분석한 관련 연구이고, 제 3장에서는 디렉토리 정보를 분류하였으며, 제 4장에서는 디렉토리 캐시 메카니즘을 설계하였고, 제 5장에서는 제안

한 기법과 타 메카니즘을 비교 평가하였고, 마지막으로 제 6장은 결론으로 구성된다.

Ⅱ. 관련 연구

분산 디렉토리 시스템에서는 사용자에게 정보의 유용성을 제공하고 성능 및 신뢰도 향상 등을 통하여 현재의 서비스 수준을 향상시켜 주어야 할 필요성이 있다. 이러한 필요성은 각각의 엔트리 및 동작에 관련된 정보의 복제를 허용하므로써 만족시킬 수 있다. X.525 권고 안에서 분산 디렉토리 시스템의 복제 메카니즘은 마스터/슬래이브 모델을 권고하고 있다. 즉, 분산 환경에서 성능을 향상시키기 위해서 복제 메카니즘을 도입하는 것이기 때문에 판독 요구는 복사 본을 갖고 있는 슬래이브 DSA에서 그대로 수용하고, 쓰기 요구에 대해서만 원본을 갖고 있는 마스터 DSA에서 수행하도록 하며, 쓰기 동작이 수행되는 동안에 발생할 수 있는 정보의 잠정적인 불일치 현상을 수용하고 있다. 그러나 이같은 복제 방법은 성능 향상에는 기여하나, 마스터 DSA의 과부하와 보안 문제, 부담료 문제 등의 어려움이 제기되고 있다[1].

분산화일 시스템인 Sprite 시스템에서는 모든 화일 서버의 디렉토리 구조를 통합한 이름 해결(name resolution) 프로토콜을 제안하였으며, 클라이언트는 서버의 이름과 주소를 매핑하는 주소 테이블을 캐시에 저장하므로써 이름 해결의 성능 향상을 꾀하였다. 그러나 수정, 추가 등의 동작에 대한 수정 전파에 대한 부담을 줄이지는 못하였다[4].

여러 개의 복사본을 유지하는데 요구되는 과부하로 초래되는 비용의 감소로 사용자 질의의 응답 시간을 개선하기 위해 캐싱 기법들이 연구되고 있다[3]. 캐싱 기법을 일관성 유지를 기준으로 나누어 보면, 강 일관성 유형과 약 일관성 유형으로 볼 수 있다. 강 일관성 기법은 캐시된 값의 일관성을 유지하기 위해 원본이 수정되면 각 복사본도 수정되어야 하는 전파(propagation) 과정이 필요하며, Mariposa 시스템에서는 규칙을 기반(rule-based)으로 수정 전파 기법을 보여주고 있다. 약 일관성 기법은 캐시된 값이 중앙 값과의 일시적인 불일치를 허용하는 기법으로, 캐시된 정보를 힌트로서 취급하므로써 분산 시스템에서의 강력 캐시 일관성을 회피하는 접근 방법[6]과 제한된 방법으로 원본과 복사본 사이의 불일치를 허용하는 Quasi 캐싱을 보여 준다[5]. 그러나 강 일관성 기법은 수정시 전파 비용이 매우 비싸며, 약 일관성 기법은 정보의 정확성을 유지하는데 어려움이 따른다.

Ⅲ. 디렉토리 정보 분류

디렉토리의 정보를 사용 특성에따라 **응용 데이타 정보, 시스템 데이타 정보, 메타 데이타 정보**로 분류하였다. 응용 데이타 정보는 사용자이름, 주소, 전화번호, 팩스번호 등과 같은 내용이 이에 해당된다. 시스템 데이타 정보는 디렉토리 관리 정보, 디렉토리 지식 정보, 부가 정보로써 디렉토리 서비스 기능을 향상시키고자 하는 정보이다. 메타 데이타 정보는 탐색 공간을 좁히기 위한 안내 정보로서 이전 질의들을 재구성한 정보이다.

3.1 캐시의 유지 기간에 따른 분류

캐시에 저장된 정보를 얼마나 오랫동안 유지할 것이냐에 따라 **장기간 정보**와 **단기간 정보**의 두 가지 영역으로 분류하였다. 이와 같은 분류는 캐시에 저장된 정보를 대체할 때 필요한 기준인 네트워크 구성상의 거리와 귀환시간을 캐시된 정보의 가중치로 반영하기 위한 분류이다. 즉, 원거리 정보와 귀환 시간이 긴 정보를 캐시에 오랫동안 유지하여 근거리 또는 귀환 시간이 짧은 정보 보다는 원거리 또는 귀환 시간이 긴 정보의 히트율을 증가시키므로써 통신망 상의 과도한 트래픽을 회피한다.

3.2 캐시의 일관성 유지에 따른 분류

디렉토리 정보를 캐시 되어진 저장 정보에 대해 일관성을 강력하게 유지하느냐 하지 않느냐에 따라 동

표 1. 일관성 유지에 따른 정보 분류
Table 1. Classification according to the consistency of cache

분류	내 용	비 고
동적 정보	- 호스트에 입출력되는 네트워크 트래픽 정보 - 통계정보 - 이력, 경향 정보	강 일관성 유지
정적 정보	- 네트워크 요소의 H/W 구성 요소 - 네트워크 요소의 S/W 구성 요소	약 일관성 유지

적 정보와 정적 정보로 분류하여 표 1에 나타내었다.

Ⅳ. 디렉토리 캐시 메카니즘

사용자는 수많은 DSA를 탐색하는 명명 서비스 질의를 위해 많은 시간을 기다릴 수는 없다. 여러 DSA 간의 통신을 줄이기 위하여, 본 절에서는 질의와 질의 처리에 대한 응답을 캐시에 저장하여 이후의 동일한 질의에 대해 수많은 DSA 접근을 피한다. 또한, 탐색공간의 부분 집합으로 질의를 고립화시키기 위해 메타 데이타 캐시에 이전 질의를 재구성한 메타 데이타 트리를 사용하는 캐시 메카니즘을 보여준다. 먼저 캐시의 종류와 저장 정보의 종류를 살펴보면 표 2와 같다.

표 2. 캐시의 종류와 저장 정보
Table 2. Stored information of cache according to types of caches

캐시 종류	저장 정보
응용데이타 질의 캐시, 응용데이타 캐시	응용데이타 정보, 시스템데이타 정보(정적 정보)
시스템데이타 질의 캐시	시스템데이타 정보(동적 정보)
메타데이타 캐시	메타데이타 정보

4.1 응용 데이타 캐시 메카니즘

응용 데이타 캐시 메카니즘은 질의의 성능을 향상시키기 위해 질의와 질의에 대한 응답을 가까운 미래에 재사용 하기 위해 응용 데이타 질의 캐시와 응용 데이타 캐시에 저장한다. 응용 데이타 질의 캐시에 저장될 내용으로는 질의 정보뿐만 아니라 질의를 유지하는 기준을 결정하기 위한 정보도 함께 저장한다.

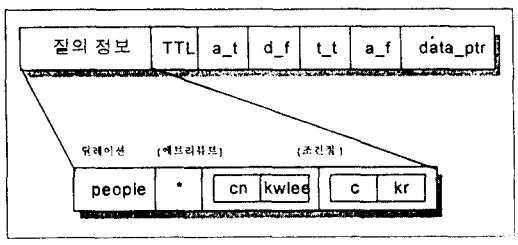

그림 1. 응용 데이타 질의 캐시의 저장 구조
Fig. 1 Stored structure of application data query cache

예를 들어, "select * from people where cn = 'kwlee' and c = 'kr'" 이라는 질의가 발생하면 그림 1과 같은 엔트리가 응용 데이타 질의 캐시에 저장된다. 그림 1에서 a_t(alive time)는 각 엔트리에 주어지는 일괄적인 할당 시간이다. d_f(distance factor)는 질의에 대한 결과를 탐색하는 과정에 대한 정보로서 네트워크 구성상의 위치 정보인 탐색 경로의 체인 수이며, t_t(turn-around time)은 질의의 귀환 시간을 시간 수치로 변환한 값이다. a_f(access frequency)는 접근 회수를 의미하며, 새로운 질의에 대해 캐시 히트가 일어났을 경우 증가하는 값이다. data_ptr은 실제 데이타(질의에 대한 응답)를 저장하고 있는 응용 데이타 캐시의 주소를 의미한다. 응용 데이타 캐시에 저장되는 내용은 그림 4와 같은 질의에 대한 결과가 저장된다.

캐시 정보의 대체 전략은 생존시간(Time To Live: TTL)을 기준으로한 최소-TTL 대체 기법(least-TTL replacement technique)을 사용하였으며, TTL의 계산 방법은 다음과 같다.

$$TTL = a\_t + 가중치-1(d\_f) + 가중치-2(t\_t) + a\_f \quad (1)$$

$$a\_t : 1 \quad (2)$$

가중치 $-1(d\_f)$:

$$\text{if } \sum_{i=1}^{n} Q_i(d\_f)/n < Q_{current}(d\_f) \text{ then } 1$$

$$\text{if } \sum_{i=1}^{n} Q_i(d\_f)/n > Q_{current}(d\_f) \text{ then } 0 \quad (3)$$

가중치 $-2(t\_t)$:

$$\text{if } \sum_{i=1}^{n} Q_i(t\_t)/n < Q_{current}(t\_t) \text{ then } 1$$

$$\text{if } \sum_{i=1}^{n} Q_i(t\_t)/n > Q_{current}(t\_t) \text{ then } 0 \quad (4)$$

$$a\_f : 히트시\ 증가값 \quad (5)$$

식(3)에서 $Q_i(d_f)$는 i 번째 질의의 거리 정보 값을 의미하며, $Q_{current}(d_f)$는 현재 질의의 거리 정보 값을 의미한다. 식(4)에서 $Q_i(t_t)$는 i 번째 질의의 귀환 시

간을 의미하며, $Q_{current}(t_t)$는 현재 질의의 귀환 시간을 의미한다. 식(1)에서 a_t는 캐시에 저장되는 모든 엔트리에 할당되는 값으로 1을 할당한다. 식(1)의 가중치 -1은 이전 질의들의 거리 정보 평균을 기준으로 결정하고, 식(2)의 가중치 -2는 이전 질의들의 귀환 시간 평균을 기준으로 결정한다. 식(1)의 a_f는 캐시에 저장된 정보가 히트시 마다 증가하는 값으로 5장의 시뮬레이션을 통하여 결정한다(트래이스 시뮬레이션 결과 t_t 대 a_f의 비율은 10 대 1일 경우가 가장 우수). 결국, 캐시에서 대체되는 엔트리는 TTL이 최소인 엔트리가 제거되는 것으로 알고리즘은 그림 2와 같다. 캐시에 저장된 엔트리의 TTL 갱신 주기는 캐시에 새로운 정보가 저장될 때 모든 엔트리의 TTL을 새롭게 계산하여 갱신한다.

```
Boolean Application_Cache_Replacement()
{
if(CACHE_FULL) then
 { repeat i
  TTL(i) = cacheRec(i).a_t+WEIGHT1(cacheRec
  (i).d_f)+WEIGHT2(cacheRec(i).t_t)+cacheRec(i).a_f
  until(i>n) /* end of cacheRec */
  DELETE(MIN(cacheRec(TTL)))
 }
STORE_CACHE(newCacheRec)
EXIT
}
```

그림 2. 응용 데이타 캐시 대체 알고리즘
Fig. 2 Replacement algorithm of application data cache

4.2 시스템 데이타 캐시 메카니즘

시스템 데이타 질의 캐시에 저장된 데이타는 강력한 일관성이 유지되어야 하므로 캐시된 질의 정보를 이용하여 새로운 접근을 수행한다. 즉, 캐시에 질의의 결과를 저장하지 않고, 질의에 대한 정보만 저장하여 히트되었을 경우에는 해당 엔트리의 접근점을 이용하여 새로운 접근을 수행하게 된다.

엔트리의 내용은 응용 데이타 질의 캐시에 저장되는 내용과 유사하나 시스템 데이타 질의 캐시에는 질의 결과를 가리키는 포인터 대신 엔트리의 정보가 포함된 DSA 접근점(a_p)을 가지고 있어 해당 질의가 히트되었을 경우에는 접근점을 이용하여 직접 접근을 수행하게 된다. 시스템 데이타 캐시의 대체 알고리즘은 응용 데이타 캐시의 대체 알고리즘과 유사하다.

4.3 메타 데이타 캐시 메카니즘

메타 데이타 캐시에는 탐색 공간의 범위를 줄이므로써 DSA의 접근 횟수를 감소시키기 위한 안내 정보를 저장한다. 탐색 공간의 결정은 새로운 질의가 발생했을 경우 이전 질의들과의 유사성을 조사하여 좀 더 직접적인 DSA 접근을 유도한다. 이를 위해서는 이전 질의들을 메타 데이타 트리로 재구성하여 메타 데이타 캐시에 저장한다. 메타 데이타 트리의 저장 구조는 그림 3과 같고, 구성은 트리 형태를 유지한다.

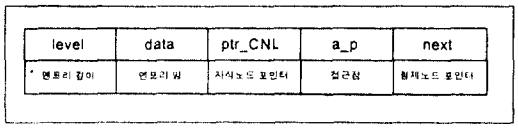

그림 3. 메타 데이타 캐시의 저장 구조
Fig. 3 Stored structure of meta data cache

메타 데이타 트리의 각 노드는 DIT 상의 엔트리 레벨(예:country, organization, 등), 엔트리 명, 하위 엔트리를 가리키는 자식 노드 포인터, 엔트리가 속한 DSA 접근점, 그리고 같은 레벨에 있는 엔트리를 가리키는 형제 노드 포인터로 구성한다.

4.3.1 메타 데이타 트리의 탐색

메타 데이타 캐시에 저장된 메타 데이타 트리의 탐색 과정을 통해 탐색 공간을 결정하는 과정을 다음의 질의를 이용하여 살펴보면 다음과 같다.

질의 1 : select * from people where cn = 'kwlee' and c = 'kr'

질의 2 : select * from people where cn = 'hschung' and o = 'hongik' and c = 'kr'

질의 1이 발생하였을 경우에는 초기 상태이므로 데이타 캐시(응용 데이타 질의 캐시 또는 시스템 데이타 질의 캐시)와 메타 데이타 캐시는 비어 있는 상태이다.

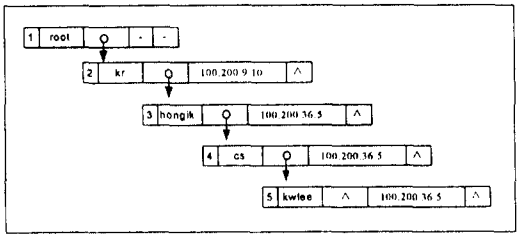

그림 4. 메타 데이타 트리의 탐색 예
Fig. 4 Search example of meta data

따라서, X.500의 SEARCH 기능을 통해 수많은 DSA들을 탐색하여 질의에 대한 응답을 얻을 것이다. 질의 1의 응답을 얻었다고 가정하면, 먼저 질의에 대한 응답을 질의를 요청한 사용자에게 보여주고, 질의가 응용 데이타 질의인지 시스템 데이타 질의인지를 판별하여 응용 데이타 질의인 경우에는 응용 데이타 질의 캐시에 질의를 저장하고 질의에 대한 응답은 응용 데이타 캐시에 저장되고, 시스템 데이타 질의인 경우에는 질의를 시스템 데이타 질의 캐시에 저장한다. 또한 메타 데이타 캐시에는 그림 4와 같이 질의 1에 대한 정보를 메타 데이타 트리로 재구성한다. 이러한 상태에서 질의 2가 도착하면, 응용 데이타 질의 캐시와 시스템 데이타 질의 캐시에 동일한 질의가 있는지를 검색한다. 이 경우에는 캐시 미스이므로 메타 데이타 캐시에 저장된 메타 데이타 트리를 이용하여 탐색 공간을 결정하여야 한다. 질의 2의 탐색 공간은 메타 데이타 트리의 탐색 알고리즘을 통하여 탐색 공간이 "o=hongik" 임을 알 수 있다. 따라서, 질의 2는 메타 데이타 캐시에 있는 "hongik"의 접근점인 "100.200.36.5"를 가지고 직접 "hongik" 엔트리가 포함된 DSA로 접근할 수 있게 된다. 메타 데이타 트리의 탐색 알고리즘은 그림 5와 같다.

```
SEARCH_DIT_SNODE(query)
/* structure ( c, o, ou, p ) query */
if DIT_SNODE.ptr_CNL = NULL then return
else if SEARCH_LIST(query.c) = FALSE then return
  else DIT_SNODE := CURRENT_LIST.CHILD
    if SEARCH_LIST(query.o) = FALSE then
      DATA_ACCESS(DIT_SNODE.cacheAddr)
      else DIT_SNODE := CURRENT_LIST.CHILD
       if SEARCH_LIST(query.ou) = FALSE then
         DATA_ACCESS(DIT_SNODE.cacheAddr)
return
END SEARCH_DIT_SNODE
```

그림 5. 탐색 알고리즘
Fig. 5 Search algorithm

4.3.2 메타 데이타 트리의 추가

질의가 발생하여 결과를 얻으면 질의와 질의에 대한 응답은 데이타 캐시에 저장되고, 질의에 대한 메타 데이타는 메타 데이타 트리에 추가되어 가까운 미래에 재사용될 정보로 재구성되어야한다. 메타 데이타 캐시의 추가 과정을 다음의 질의를 이용하여 살펴보면 다음과 같다.

질의 1 : select * from people where cn = 'kwlee' and c = 'kr'

질의 3 : select * from people where cn = 'jhlee' and o = 'hongik' and c = 'kr'

질의 1의 결과가 그림 4와 같이 이미 메타 데이타 트리에 생성된 상태에서 질의 3의 응답이 도착하면 메타 데이타 트리에 추가 알고리즘을 통하여 새로운 노드로 추가 되어야 한다. 추가할 경우에는 기존 메타 데이타 트리와 매핑 과정을 통하여 메타 데이타 트리에 새로운 노드로 추가되어 진다. 그림 6은 질의 3에 대한 정보가 추가되어 재구성된 결과이다. 메타 데이타 트리의 추가 알고리즘은 그림 7과 같다.

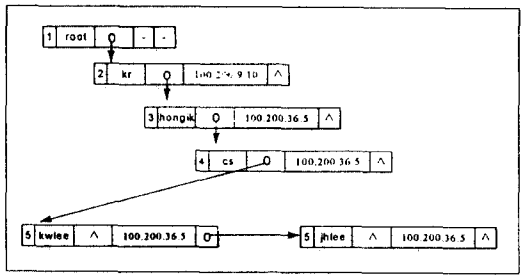

그림 6. 메타 데이타 트리의 추가 예
Fig. 6 Example of adding meta data cache

```
ADD_NODE(Q_result, LEVEL)
if LEVEL >= 4 return
if DIT_SNODE.ptr_CNL = NULL then
                    ADD_LIST(Q_result[LEVEL])
else if SEARCH_LIST(Q_result[LEVEL]) = FALSE
                then ADD_LIST(Q_result[LEVEL])
    else LEVEL := LEVEL + 1
        DIT_SNODE := CURRENT_LIST.CHILD
        ADD_NODE(Q_result, LEVEL)
return
END ADD_NODE
Func: ADD_DIT_SNODE(Q_result)
    Q_result[4][MAX_LENGTH], DIT_SNODE
    LEVEL := 0
    ADD_NODE(Q_result, LEVEL)
return
END ADD_DIT_SNODE
```

그림 7. 추가 알고리즘
Fig. 7 Add algorithm

4.3.3 메타 데이타 트리의 노드 삭제

메타 데이타 트리에서의 삭제는 데이타 캐시에서 제거되는 엔트리에 대한 질의정보를 메타 데이타 트리에서 삭제한다. 메타 데이타 캐시에 그림 8과 같이 메타 데이타 트리가 저장되어 있고, "select * from people where cn = 'jhlee' and o = 'etri' and c = 'kr'"인 질의가 데이타 캐시에서 제거되어진다면 형제 노드가 있는 노드를 찾아 노드의 링크를 삭제한다. 메타 데이타 트리의 삭제 알고리즘은 그림 9와 같다.

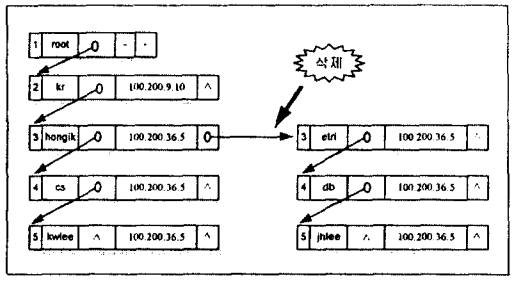

그림 8. 메타 데이타 트리의 삭제 예
Fig. 8 Example of deleting meta data tree

```
DELETE_DIT_SNODE(entry, LEVEL,DELETED)
/* entry는 cache에서 대치되는 정보 */
if LEVEL >= 4  or DELETED = TRUE
                        return(DELETED)
SEARCH_LIST(entry[LEVEL])
DIT_SNODE := CURRENT_LIST.CHILD
if COUNT(DIT_SNODE.ptr_CNL) = NULL then
                        return(DELETED)
else if COUNT(DIT_SNODE.ptr_CNL) > 1  then
        FLAG := 0 else FLAG := 1
LEVEL := LEVEL + 1
DELETE_DIT_SNODE(entry,LEVEL)
if FLAG = 0 then { DELETED:=TRUE,
                        return(DELETED) }
else DELETE_NODE(entry[LEVEL])
END DELETE_DIT_SNODE
```

그림 9. 삭제 알고리즘
Fig. 9 Delete algorithm

4.4 질의 처리 구성

그림 10은 효율적인 질의 처리를 위한 구성이다. 질의가 도착하면 먼저 질의 관리자는 캐시 관리자를 통하여 도착한 질의가 응용 데이타 질의 인지 시스템 데이타 질의 인지를 결정한 후 동일한 질의가 데이타 캐시에 있는지를 검사한다. 검사 결과 적중(hit) 한다면, 응용 데이타 질의인 경우는 응용 데이타 캐시에 저장된 정보를 그대로 사용자에게 보여주고, 시스템 데이타인 경우에는 질의 처리기를 통하여 캐시되어진 정보에서 접근점을 가지고 새로운 디렉토리 접근을 수행하게 된다. 만약 새로운 질의가 캐시 미스라

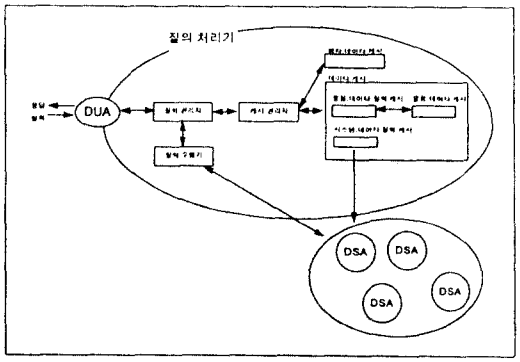

그림 10. 질의 처리 구성
Fig. 10 Structure of query processing

면 메타 데이타 캐시에 있는 메타 데이타 트리를 사용하여 질의의 탐색공간을 결정한다.

탐색공간의 결정은 메타 데이타 트리의 탐색 알고리즘을 통하여 결정하고, 결정된 탐색 공간의 접근점을 이용하여 해당 DSA로 직접 접근하게 된다. 이러한 과정을 통하여 얻어진 새로운 질의에 대한 응답은 가까운 미래에 재사용을 위하여 데이타 캐시에 저장된다. 또한 질의를 메타 데이타 트리에 추가하여 재구성 한다.

V. 성능 평가

본 장에서는 제안 알고리즘의 우수성을 입증하기 위하여 다음의 두가지 방법으로 성능을 평가 하였다. 첫째, 기존의 대체 알고리즘과의 성능평가로서 기존에 가장 많이 사용되는 LRU(Least Recently Used)와 LFU(Least Frequently Used) 알고리즘과의 성능 평가이다. 둘째, 메타 데이타 사용 여부에 따른 성능 평가로서 본 논문에서 제안한 메타 데이타 캐시의 사용과 X.500에서의 방법과 성능 평가를 하였다.

5.1 캐시의 대체 전략 비교

(1) 비교 대상 기법 선정

기존의 대체 알고리즘 중 대표적인 4가지 기법을 살펴보면 다음과 같다. 최적(optimal) 기법은 이상적이지만 구현이 어렵고, LRU 기법은 가장 오랜 동안 사용되지 않은 엔트리를 대체하는 방법이며, LFU 기법은 참조 빈도수가 가장 낮은 엔트리를 대체하는 방법이다. 마지막으로, 무작위(random) 기법은 특정한 전략없이 대체 알고리즘을 수행하므로 성능이 낮다. 따라서 본 논문에서는 비교 대상 알고리즘으로 LRU와 LFU를 선정한다.

(2) 성능 평가

성능측정에 사용될 시뮬레이션 방식은 트레이스-유추(trace-driven) 방식이며, 사용되는 트레이스는 스프라이트 트레이스(sprite trace)[7]를 사용한다. 스프라이트 트레이스는 분산 환경의 접근형태를 알기위하여 버클리 대학에서 만든 트레이스이다. 이 트레이스는 특수한 형태로 바꾸어 압축된 형태로 제공한다. 따라서, 스프라이트 트레이스와 함께 제공하는 라이브러리를 사용하여 필요한 인자를 얻어 시뮬레이션

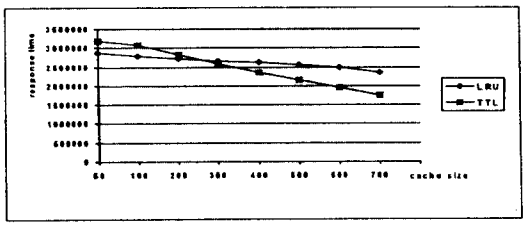

그림 11. 제안된 알고리즘과 LRU 알고리즘의 성능분석
Fig. 11 Performance result of proposed algorithm and LRU algorithm

하였다.

그림 11은 최소-TTL 대체 기법과 LRU 기법을 성능분석한 결과이며 특히, 캐시의 크기가 600일 경우 최소-TTL 대체 기법이 LRU 기법보다 22%의 성능향상을 보여준다. 그림 12는 최소-TTL 대체 기법과 LFU 기법을 성능 분석한 결과이다.

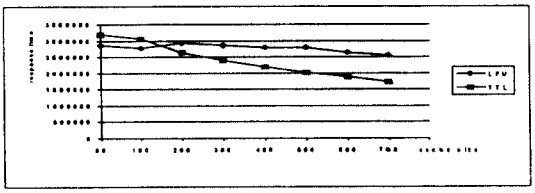

그림 12. 제안된 알고리즘과 LFU 알고리즘의 성능분석
Fig. 12 Performance result of proposed alogorithm and LFU algorithm

5.2 메타 데이타 캐시 사용 비교

예제 질의를 처리하기 위한 분산 디렉토리 환경은 DSA가 "organization(o)" 레벨에 있다고 가정하고, "country(c)", "locality(l)", "organization"의 수가 표 6과 같이 단계별로 변하고, 메타 데이타를 사용할 때와 사용하지 않을 때의 각 질의에 대한 평균 DSA 접근 횟수에 대한 비교를 그림 13과 그림 14에 나타내었다. 그림 13과 그림 14에서 보는 바와같이 특히 질의 6, 7, 8의 경우는 X.500의 객체 탐색보다 메타 데이타를 사용한 모델의 객체 탐색을위한 DSA 접근횟수가 현저히 감소하는 것을 볼 수 있다.

질의 1 : select * from people where cn = "kwlee" and

c = "kr" ;

질의 2: select * from people where cn = "hschung" and c = "kr" ;

질의 3: select * from people where cn = "hschung" and l = "seoul" and c = "kr" ;

질의 4: select * from people where cn = "kwlee" and l = "seoul" and c = "kr" ;

질의 5: select * from people where cn = "kwlee" and l = "pusan" and c = "kr" ;

질의 6: select * from people where cn = "kwlee" and o = "pwu" and c = "kr" ;

질의 7: select * from people where cn = "hschung" and o = "hongik" and c = "kr" ;

질의 8: select * from people where cn = "psshin" and o = "hongik" and c = "kr" ;

표 3. 단계별 DSA 수 변화
Table 3. Number of DSA by step

엔트리	단계				
	1	2	3	4	5
Country	1	1	1	1	1
Locality	10	20	30	40	50
Organization	10	20	30	40	50
총 계	100	400	900	1600	2500

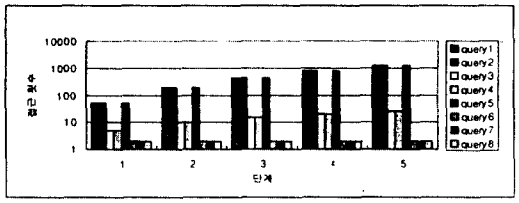

그림 13. 제안된 모델에서의 성능 분석
Fig 13 Performance result in proposed model

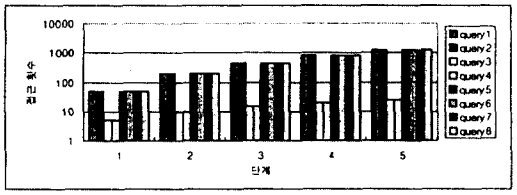

그림 14. X.500에서의 성능분석
Fig. 14 Performance result in X.500

Ⅵ. 결 론

본 논문에서는 분산 디렉토리 환경에서 질의 처리 속도를 높이기 위해 원격지의 객체에 대한 질의와 결과를 요청지의 캐시에 저장하는 캐싱 메카니즘을 설계하였다. 주요 특징을 요약하면 다음과 같다.

첫째, 분산 디렉토리 시스템에서 저장·관리하는 정보를 응용 데이타, 시스템 데이타, 메타 데이타 정보로 분류하였다. 둘째, 분류된 정보를 장기간 정보, 단기간 정보, 동적 정보, 정적 정보로 분류하는 캐시의 유지 기간과 캐시의 일관성 유지 기준을 생성하였다. 셋째, 분류된 캐시 정보를 저장하는 캐시 시스템 구조와 저장구조를 설계하였다. 넷째, 데이타 캐시(응용 데이타 캐시, 시스템 데이타 캐시)의 대체 알고리즘으로 거리 정보와 접근 회수를 가중치로 부여한 최소-TTL 대체 알고리즘을 제안하였다. 다섯째, 질의에 대한 탐색 공간의 범위를 좁힘으로써 질의 속도를 향상시키기 위해 이전 질의를 재구성한 메타 데이타 트리를 저장하는 메타 데이타 캐시의 운영 알고리즘을 제안하였다. 마지막으로, 제안된 캐시 메카니즘(TTL)과 LRU 메카니즘 LFU 메카니즘과의 성능 평가를 수행하여 성능 향상을 입증하였으며, X.500의 객체 탐색보다 메타 데이타를 사용한 제안된 모델에서 DSA 접근횟수가 현저히 감소하는 것을 보였다.

향후 연구 분야로는 본 논문의 접근 횟수 계산 환경을 실제 환경과 동일한 조건으로 접근 횟수 계산을 수행하므로써 더욱 정확한 결과 값을 얻도록하며, 디렉토리 스키마의 동적 환경 변화에 대한 지원 정책 개발도 연구되어야 한다.

참 고 문 헌

1. CCITT, *The Directory : Recommendations X.500, X.501, X.509, X.511, X.518, X.519, X.520, X.521, X.525*, CCITT Blue Book, 1991.
2. 이재호, *통신망 환경하에서 객체-능동-지식 기반 디렉토리 데이타베이스 모델 설계*, 홍익대학교, 박사학위논문, 1996. 8.
3. Jean-Chrysostome Bolot, Hossam Afifi, *Evaluating Caching Schemes for the X.500 Directory System*, The 13th ICDCS, Pittsburgh, Pennsylvania, May

25-28, 1993, pp. 112-119.
4. Ordille, J.J. *Descriptive Name Services for Large Internets*, Ph.D. Thesis, University of Wisconsin, Nov. 1993.
5. R. Alonso, D. Barbara, H. Garcia-molina, *Data Caching Issues in an Information Retrieval System*, ACM Transactions on Database Systems. Vol. 15, No. 3, Sept. 1990, pp. 359-384.
6. D. B. Terry, *Caching Hints in Distributed Systems*, IEEE Transactions on Software Engineering, Vol. SE-13, No. 1, Jan. 1987, pp 48-54.
7. J. H. Hartman, *Using the Sprite File System Trace*, Berkeley University, 1993.
8. 이강우, 이재호, 임해철, 분산 *디렉토리 시스템을 위한 캐시 메카니즘*, 한국정보과학회 '96 가을 학술발표집(A), 제23권, 2호, pp. 213-216, 10. 1996.

李 康 雨(Kang Woo Lee) 정회원
1987년 : 홍익대학교 전자계산학과(이학사)
1989년 : 건국대학교 전자계산학과(이학석사)
1992년~현재 : 홍익대학교 전자계산학과 박사과정
1994년~현재 : 서남대학교 전산정보학과 전임강사
※주관심분야 : 분산 데이타베이스, 프로토콜 공학, 객체지향 데이타베이스

李 載 昊(Jae Ho Lee) 정회원
1987년 : 홍익대학교 전자계산학과(이학사)
1989년 : 홍익대학교 전자계산학과(이학석사)
1996년 : 홍익대학교 전자계산학과(이학박사)
1989년~1996년 : 한국정보통신연구소 선임연구원
1996년~현재 : 인천교육대학교 컴퓨터교육과 전임강사
※주관심분야 : 컴퓨터교육, 분산 데이타베이스, 프로토콜 공학

林 海 喆(Hae Chull Lim) 정회원
1976년 : 서울대학교 계산통계학과(이학사)
1978년 : 한국과학기술원 전자계산학과(이학석사)
1988년 : 서울대학교 컴퓨터공학과(공학박사)
1978년~1981년 : 현대엔지니어링
1989년~1990년 : 미국 플로리다대학 방문교수
1981년~현재 : 홍익대학교 컴퓨터공학과 교수
※주관심분야 : 객체지향 데이타베이스, 실시간 데이타베이스, 분산 데이타베이스