# 분산 디렉토리 시스템에서의 원격 질의에 대한 캐싱 기법

이 강 우†


## 요 약

본 논문에서는 분산 디렉토리 시스템에서 질의 처리 속도를 향상시키기 위하여 원격지의 객체에 대한 질의와 결과를 요청지의 캐시에 저장하는 캐싱 메카니즘을 제안한다. 이를 위하여 첫째, 분산 디렉토리 시스템에 저장되는 캐시 정보를 응용 데이터 정보, 시스템 데이터 정보로 분류하고, 분류된 캐시 정보를 기반으로 캐시 시스템 구조를 설계하였다. 둘째, 각각의 캐시에 대한 거리 정보와 접근 회수를 가중치로 부여한 최소-TTL 대체 기법을 개발하였다. 마지막으로 제안된 캐시 기법과 타 기법(LRU, LFU 대체 기법)에 대하여 성능 평가를 수행하여 제안된 기법이 LRU 기법 보다 25%의 성능향상을 보였으며, LFU 기법보다는 30%의 성능향상 결과를 보였다.


# A Caching Mechanism for Remote Queries in Distributed Directory Systems

Kang-Woo Lee†


## ABSTRACT

In this paper, for improving the speed of query processing on distributed directory system, we proposed a caching mechanism which is store the queries and their results on the remote site objects in the cache of local site. For this, first, cached information which is stored in distributed directory systems is classified as application data and system data. And cache system architecture is designed according to classified information. Second, least-TTL replacement mechanism which uses the weighted value of geographical information and access frequency for replacements are developed for each cache. Finally, performance evaluations are performed by comparing the proposed caching mechanism and other mechanisms (LRU, LFU replacement). Our least-TTL mechanism shows a performance improvement of 25% over the LRU and that of 30% over LFU.


## 1. Introduction

As the amount of information that is of interest to users is increased exponentially due to the increase in complexity and size of communication networks, the need for distributed information repository is requested to efficiently manage the large amount of distributed data. One of the technologies to meet the needs of such requirements is the ITU's(International Telecommunication Union) X.500 directory system [6].

The X.500 directory system is different from a general-purpose DBMS in several aspects and thus can be considered as a special-purpose DBMS. In a

† 정 회 원 : 서남대학교 컴퓨터정보통신학부 교수
논문접수 : 1999년 5월 4일, 심사완료 : 1999년 11월 2일

distributed directory system, the stored data is static in nature with little or no modification.

The information held in the directory is called the Directory Information Base(DIB). The DIB consists of entries(or objects) which contain information about entities. Entities in the DIB are represented by entries in a global, hierarchical name space called the Directory Information Tree(DIT). Each object belongs to at least one object class. The class determines the attributes that can be present in the object. Each attribute in an object is composed of a type and one or more values. For naming service which searches an object from distributed directory throughout the world, we must understand the whole organization of the communication network and estimate a unique path name of the object or navigate all paths. But it is difficult for users to know the organization of the communication network or path names of the objects in which he/she is interested, and if we do not know the path name of the object, one must search many data repositories. In order to solve the above problem, most current distributed systems use descriptive naming service [9]. But descriptive naming service has low performance. "SEARCH" of X.500 is the most commonly used descriptive naming service and global selection query. In order to get the results of a global selection query, the user must search thousands of databases [6]. The solution to low query performance is based on the concept that the cost for searching a local name can be reduced by using name caching. But in X.525 recommendation standards, only the data replication scheme is provided without including the data cache scheme.

## 2. Related Works

The distributed directory system needs to improve the present service level by providing information usability, performance improvement and high conviction. This is satisfied with allowing the duplication of each entry and operational information [6]. The replication mechanism of the distributed directory system in recommendation of X.525 recommends master/slave model. This master/slave model is composed of primary and secondary shadowing methods. In primary shadowing, master DSA is unique replication information provider, and slave DSA is able to read, compare, search and open. All of the modification operations are performed by master DSA. In the secondary shadowing, master DSA is not unique replication information provider, and some authorized DSAs give modified information to master DSA. Such a replication method could improve the system performance, but there are some problems, such as overload of the master DSA, security and so on.

Many of caching mechanisms improving response time for user query by reducing overload cost for holding replications have been studied [1]. The Mariposa system shows a rule-based modification propagation method. As strong cache consistency avoidance methods weak consistency method treats cache information as hint, and Quasi caching permits an inconsistency manner [10]. But the modification propagation cost of strong consistency method is very high, and accuracy management of weak consistency method is difficult.

Therefore, this paper designs the caching mechanism which avoids replication management overload and holds consistency according to the characteristics of data. Based on classified cache information, we design cache system structure and storage structure of each cache information, and develop a cache management algorithm for each cache information.

## 3. Classification of directory information

In this section, for designing an effective caching mechanism, we analyze the directory information which is maintained and stored in distributed directory systems.

- Classification according to the properties of directory information

  According to the properties of directory infor-

mation, we classify cache information into application data information, and system data information.

i ) Application data information
- directory user information

ii ) System data information
- directory managing information
- directory knowledge information
- spplementary information

• Classification according to the life time of cache

According to the life time of cache, we classify cache information into long-term information and short-term information. When cache replacement is required, this classification is used to apply a weighted value of network distance and turn around time. Information of long distance and long turn around time is maintained in the cache longer than that of short distance and short turn around time. It reduces heavy traffic of the communication network by improving the hit ratio of long distance and long turn around time information respectively.

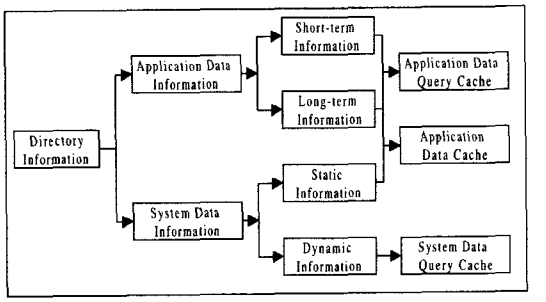
• Classification according to the consistency of cache

According to the properties of system data information was maintained in strong consistency or maintained weak consistency, we classify cache information into dynamic information and static information.

i ) Dynamic information
- network traffic information
- statistics information
- historical data, trend data
- hard-time constraints data

ii ) Static information
- hardware component of network
- software component of network
- network information for component itself
- soft-time constraints data

(Fig. 1) represents storage caches for classified information by three classification methods. Especially, dynamic information, which is stored in the system data query cache, is stored for maintaining strong consistency. Maintaining the strategy of cache information is basically weighted by geographical information, turn-around time and access frequency. It's a strategy for the informations of long distance and turn-around time to maintain in the cache longer than other informations. So this strategy can avoid overhead traffic of long-term information by increasing the hit ratio of long-term information over short-term information.




(Fig. 1) Storage caches for classified information

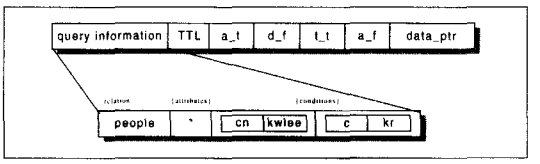
## 4. Directory caching mechanism

In this section, to reduce the communication overhead among various DSA's, the use of cache which stores repeated query and response will be discussed.

### 4.1 Application data caching mechanism

The information stored in the application data cache are composed as follows : ① application data such as user name, address, phone number, and fax numbers which identify the directory's own facilities. ② system's static data subset such as information about the network, information about network hardware and software, and weak time constraint data. The user can get the benefit of reduced query processing time from the application data caching mechanism which stores the query and their result into an application data query cache repeatedly. And not only the query contents but also some additional information which determines preservation of the query will be stored into the application data query

cache. For example, the entries stored in the application data query cache are shown in (Fig. 2) when a query is supplied as **"select * from people where cn = 'kwlee' and c = 'kr'"**.




(Fig. 2) Storage structure of application data query cache

In (Fig. 2), *a_t* is assigned to the live time of each query, *d_f* stands for the number of search path chain which describe the location of the given network configuration so that this is the information about the process of result searching. *t_t* means turn-around time and specifies query result return time. *a_f* describes how often the access has happened and may be increased each time when a cache hit occurs on a new query approval. *data_ptr* is the pointer to the address of actual response data from given query which is in the application data cache. Application data cache contents are the results from the queries such as given in (Fig. 2).

The cache information replacement strategy is based on Least-TTL replacement technique. The TTL computation is shown as follows.

(1) $TTL = a\_t + weight\text{-}1(d\_f) + weight\text{-}2(t\_t) + a\_f$

(2) $weight\text{-}1(d\text{-}f)$ :

if $\sum_{i=1}^{n} Q_i(d-f)/n < Q_{current}(d-f)$, then 1 else 0

(3) $weight\text{-}2(t\text{-}t)$ :

if $\sum_{i=1}^{n} Q_i(t-t)/n < Q_{current}(t-t)$, then 1 then 0

(4) $a\text{-}f$ : increment value when cache hit happens

$Q_i(d\text{-}f)$ in expression(2) is query distance information and $Q_{current}(d\text{-}f)$ is the distance information of the current query. $Q_i(t\text{-}t)$ in expression(3) is the query return time and $Q_{current}(t\text{-}t)$ is turn-around time of current query. *a_t* in expression(1) is a value assigned to each entry in the cache and the initial value is 1. *Weight-1* in expression(1) is determined from the average of distance information in the query supplied before, and *weight-2* in expression(1) is determined from the average turn around time of the query supplied before. *a_f* in expression(1) increases each time when a cache hit happens.

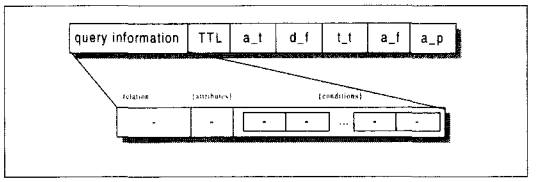
```
Boolean Application_Cache_Replacement()
{
if(CACHE_FULL) then
 { repeat i
   TTL(i) = cache_rec(i).a_t + WEIGHT1(cache_rec(i).d_f) +
      WEIGHT2(cache_rec(i).t_t) + cache_rec(i).a_f
  until(i>n) /* end of cache_rec */
  DELETE(MIN(cache_rec(TTL)))
 }
 STORE_CACHE(new_cache_rec)
 EXIT
}
```

(Fig. 3) Replacement algorithm of application data cache

Eventually, the cache entry whose TTL value is the least will be replaced and thrown out, and the full algorithm is shown in (Fig. 3). Whenever new information is added into the cache, the TTL of all the cache entries should be computed and updated.

### 4.2 System data caching mechanism

The informations stored in the system data query cache are queries about dynamic information such as network traffic information, statistics information, temporal information, trend information, the average of network load information and strong time constraint data etc. Especially, because the data stored in the system data query cache preserves strong consistency, a new access is performed using the query information cached. Namely, instead of the query result not being stored in cache, only the information about the query is stored. So, in case that hit is performed a new access is used. (Fig. 4) shows the storage structure of the entry which is stored in the system data query cache.




(Fig. 4) Storage structure of system data query cache

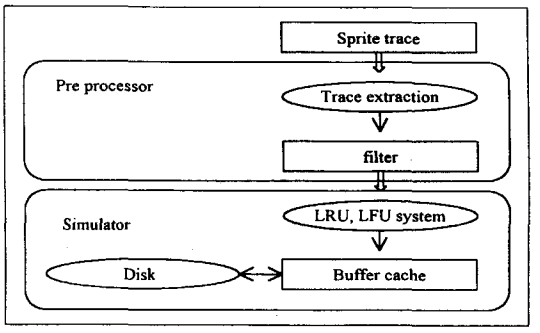
The contents of a entry are similar to those stored in the application data query cache, but the system data query cache has DSA access point(*a_p*) which includes the information of a entry instead of a pointer to the query result. So, in case the query is hit, data access is performed using the access point. The replacement algorithm of the system data cache is similar to that of the application data cache.

## 5. Performance measurements

In this section, we compare the Least-TTL technique of the data cache developed in section 4 with the most general method LRU (least recently used) technique and LFU (least frequently used).

### 5.1 Simulation model

Basically the simulation was performed through the preprocessor and the simulator. The preprocessor is separated into the extraction part of the trace and the filter part to extract all needed trace data. The entire configuration is shown in (Fig. 5). The pre processor extracts needed factors then translates for the simulator. It is performed through detail phases in the preprocessor phase. Translated logical block number transfers to the buffer cache. It is managed in the part of the buffer cache. The replacement algorithm of the buffer cache can use selectively proposed Least-TTL, LRU, and LFU techniques.



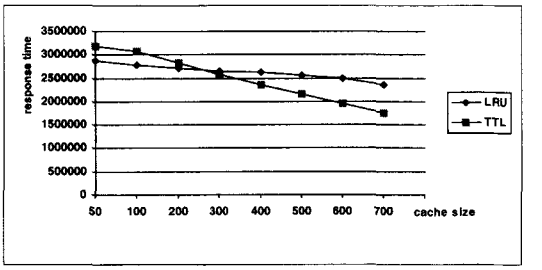

(Fig. 5) Configuration of simulation

If the request block exists in part of the buffer cache isn't returned to the physical block but just returned to a signal which exists in buffer cache. If the requested block does not exist it sends a read signal to the disk module. This request gets a response time in the disk module.

### 5.2 Comparison of replacement techniques of cache

The simulation method used in performance monitoring is a trace-driven method, and the trace data is a Sprite trace [5]. The Sprite trace was developed at Berkeley University to monitor the shape of access in a distributed environment. We simulate using both a Sprite trace and libraries provided together and retrieve the parameter needed. This simulation was experimented on SUN SPARC 20 and each module was implemented in C. And our trace data used July's trace of six traces in 1994 [5].

(Fig. 6) shows the result of the performance analysed between the Least-TTL method and LRU method. The Least-TTL method shows the performance improvement of 22% more than the LRU method.

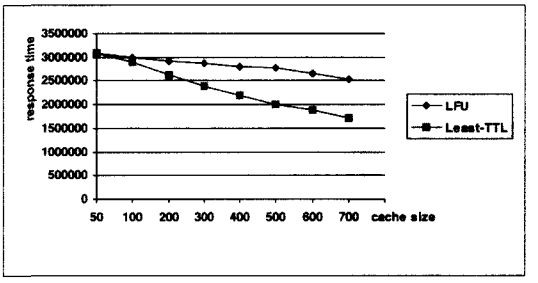



(Fig. 6) Performance result of proposed algorithm and LRU algorithm

<Table 1> Performance analysis table of proposed mechanism and LRU mechanism

| | cache size | 50 | 200 | 400 | 600 |
|---|---|---|---|---|---|
| least-TTL | LT | 1.99 | 5.57 | 10.59 | 15.48 |
| | ST | 8.21 | 9.66 | 10.73 | 10.97 |
| | HR | 10.2 | 15.23 | 21.32 | 26.45 |
| | RT | 3092567 | 2627722 | 2193425 | 1877580 |
| LRU | LT | 2 | 3.50 | 4.46 | 5.82 |
| | ST | 10.53 | 13.47 | 16.60 | 19.66 |
| | HR | 12.53 | 16.97 | 21.06 | 25.48 |
| | RT | 3095185 | 2919445 | 2799947 | 2647890 |

LT : hit ratio of long-term information
ST : hit ratio of short-term information
HR : hit ratio of total
RT : response time




(Fig. 7) Performance result of proposed algorithm and LFU algorithm

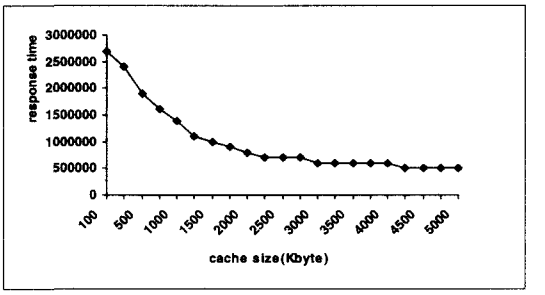
<Table 2> Performance analysis table of proposed mechanism and LFU mechanism

| | cache size | 50 | 200 | 400 | 600 |
|---|---|---|---|---|---|
| least-TTL | LT | 1.99 | 5.57 | 10.59 | 15.48 |
| | ST | 8.21 | 9.66 | 10.73 | 10.97 |
| | HR | 10.2 | 15.23 | 21.32 | 26.45 |
| | RT | 3092567 | 2627722 | 2193425 | 1877580 |
| LFU | LT | 3.77 | 4.92 | 5.58 | 6.61 |
| | ST | 16.32 | 19.42 | 21.17 | 23.84 |
| | HR | 20.09 | 24.32 | 26.75 | 30.45 |
| | RT | 2852489 | 2703099 | 2619803 | 2492057 |

(Fig. 7) shows the result of the performance analysed between the Least-TTL method and LFU method. The Least-TTL method shows the performance improvement of 35% than the LFU method. <Table 1> and <Table 2> are results of the performance analysis for the hit ratio of long-term information, the hit ratio of short-term information, the total hit ratio and response time by increasing cache size.

5.3 Cache size

For procedure optimal cache size in distributed directory environments we simulate response time using in the front section. In (Fig. 8), we show decreasing ratio of response time which reduces rapidly between 2000 and 2500. So optimal cache size is between 2000 and 2500.




(Fig. 8) Performance result of proposed algorithm and LFU algorithm

## 6. Conclusion

The paper designs a caching mechanism that improves the query performance in a distributed directory environment by storing the query and results of remote site objects in the cache query request site. The main properties are summarized as follows. First, we classify the information stored and managed in distributed directory systems as application data, and system data. Second, we develop the lifetime period in the cache and the lifetime period criteria of the cache which classifies the long-term information, short-term information, dynamic information and static information. Third, we designed the structure of the cache system and the storage structure of the classified cache information. Fourth, we propose the Least-TTL algorithm which used weighted value on the geographical information and the access frequency as a replacement algorithm of the data cache. Finally, we showed a performance improvement using the proposed caching mechanism compared with LRU

mechanism, through performance evaluation.

## References


[1] 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.

[2] Matthew Addison Blaze, Caching in Large-Scale Distributed File Systems, Ph.D. Thesis, University of Princeton, January 1993.

[3] Jean-Chrysostome Bolot, Hossam Afifi, Evaluating Caching Schemes for the X.500 Directory System, *The 13th International Conference on Distributed Computing Systems*, Pittsburgh, Pennsylvania, May 25-28, 1993, pp.112-119.

[4] James Gwertzman, Autonomous Replication in Wide-Area Internetworks, Ph.D. Thesis, University of Harvard, April 1995.

[5] J. H. Hartman, *Using the Sprite File System Trace*, Berkeley University, 1993.

[6] ITU, The Directory : Recommendations X.500, X.501, X.509, X.511, X.518, X.519, X.520, X.521, X.525, ITU Blue Book, 1991.

[7] K. W. Lee, J. H. Lee, H. C. Lim, A Cache Mechanism for Distributed Directory System, *Proceedings of the 22 KISS Fall Conference*, Vol.23, pp.213-216, 10. 1996.

[8] B. Clifford Neuman, Scale in Distributed Systems, *Readings in Distributed Computing Systems*, IEEE Computer Society Press, 1994.

[9] Ordille, J. J. Descriptive Name Services for Large Internets, Ph.D. Thesis, University of Wisconsin, Nov. 1993.

[10] D. B. Terry, Caching Hints in Distributed Systems, *IEEE Transactions on Software Engineering*, Vol.SE-13, No.1, Jan. 1987.

[11] Brent Ballinger Welch, Naming, State Management, and User-Level Extensions in the Sprite Distributed File System, Ph.D. Thesis, University of California at Berkeley, 1990.

[12] Craig Hunt, TCP/IP Network Administration, O'Reily & Associates, Inc. 1992.



**이 강 우**

e-mail : kwlee@tiger.seonam.ac.kr

1987년 홍익대학교 전자계산학과 졸업(이학사)

1989년 건국대학교 전자계산학과 졸업(공석사)

1997년 홍익대학교 전자계산학과 졸업(이학박사)

1994년~현재 서남대학교 컴퓨터정보통신학부 조교수

관심분야 : 분산 데이터베이스, 분산시스템, 객체지향 데이터베이스