분산 디렉토리 데이터베이스 시스템에서의 메타 데이터 캐싱 기법

이 강 우† · 고 진 광††

요 약

본 논문에서는 분산 디렉토리 데이터베이스에서 질의 처리의 속도를 증가시키기 위한 캐싱 기법을 제안한다. 관심 대상 객체의 탐색 시간과 질의 처리 시간을 감소시키기 위하여 원격 지에 위치한 객체에 대한 요청 질의와 요청 결과를 요청 사이트의 캐시에 저장한다. 캐시 시스템 구조는 분류된 정보에 따라 설계되고, 캐시 스키마는 각각의 캐시 정보에 따라 설계되어 있다. 각각의 캐시에 대한 운영 알고리즘은 메타 데이터 트리를 메타 데이터 캐시를 위하여 작성하였다. 메타 데이터 트리는 탐색 공간의 범위를 줄이므로서 질의 처리의 속도를 향상시킨다. 마지막으로 성능 평가는 제안된 캐시 기법과 X.500의 기법을 비교하여 수행하였다.

Meta Data Caching Mechanism in Distributed Directory Database Systems

Kang-Woo Lee† · Jin Gwang Koh††

ABSTRACT

In this paper, a cache mechanism is proposed to improve the speed of query processing in distributed directory database systems. To decrease search time of requested objects and query processing time, query requests and results about objects in a remote site are stored in the cache of a local site. Cache system architecture is designed according to the classified information. Cache schema are designed for each cache information. Operational algorithms are developed for meta data cache which has meta data tree. This tree improves the speed of query processing by reducing the scope of search space. Finally, performance evaluation is performed by comparing the proposed cache mechanism with X.500.

1. Introduction

This paper addresses the challenge of locating people, resources, and other objects in global Internet. As the large internet grows beyond a million hosts in tens of thousands of organizations, it is increasingly difficult to locate any particular object. Users locate resources by describing resource attributes. The proposed method provide fast query processing in large internet. 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, 7].

The X.500 directory system is different from a general-purpose DBMS(database management system) in several aspects and thus can be considered as a spe-

† 정 회 원 : 서남대학교 컴퓨터정보통신학부 교수
†† 종신회원 : 순천대학교 컴퓨터과학과 교수
논문접수 : 1999년 9월 6일, 심사완료 : 2000년 5월 26일

cial-purpose DBMS. In a directory system, the stored data is static in nature with little or no modification. In addition, the distributed directory system allows temporary inconsistency since the system does not require complete global commitment [1, 3].

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) [3, 6]. 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 the 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 [4, 9]. Descriptive naming service searches objects by describing attributes of object, for example, **"Select * from people where name = 'kwlee'"**, and it is possible that the user can access the object without understanding the name space structure of complex information. But descriptive naming service has low performance. "SEARCH" of X.500 is the most commonly used descriptive naming service and global selection query [8]. In order to get the result of a global selection query, the user must search thousands of databases [3, 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 [2, 5]. But in X.525 recommendation standards, only the data replication scheme is provided without including the data cache scheme.

This paper proposes a caching mechanism for storing queries of an object in remote sites and for storing their results in cache at query request sites so that query speed can be improved in distributed directory environments.

2. Related work

The distributed 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 [4, 6, 9]. The replication mechanism of the distributed directory system in recommendation of X.525 recommends master/slave model. Because replication mechanism is used to improve performance in distributed environment, read requirement is managed by slave DSA(directory system agent) where replication copies are placed, and write requirement is managed by master DSA where original data are placed. While a write operation is executed, inconsistency is temporarily permitted. In its functional aspect, DSA in master/slave model is the source of the replication information, and then acts as a master or a sender of replication information. 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 systems performance, but there are some problems, such as the overload of the master DSA, security and so on.

Sprite distributed file system suggests a name resolution protocol which combines all directory structures of file servers. The client system improves performance of name resolution by storing address tables in caches which maps the server's name to the address. But it can't decrease the load of modification propagation for modification and insertion.

Many of cache mechanisms improving response time for a user query by reducing overload cost for holding replications have been studied [1, 2]. Based on the consistency, there are strong consistent types and weak consistent types. In the strong consistency method, the source modification must be propagated into each replication. The Mariposa system shows a rule-based modification propagation method. As a strong cache consistency avoidance method, a weak consistency method treats cache information as hint, and Quasi caching permits an inconsistency manner [9, 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 uses meta data tree cache and holds consistency according to the characteristics of data. First, we classify cache information into application data information and system data information according to consistency maintenance of cache information stored in the distributed directory system, and classify previous query information into meta data information to avoid similar long distance access. 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. Directory cache mechanism

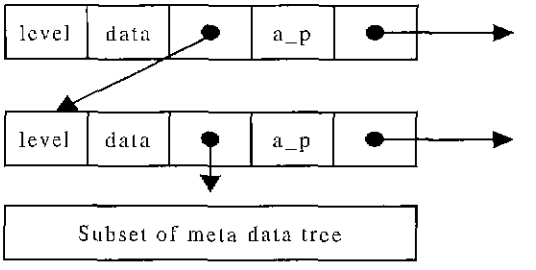
In this section, to reduce the communication overhead among various DSA's, the use of cache which stores repeated query and response will be discussed. And we will see the cache strategy which defines the storing method of the meta data tree reformed from queries previously supplied into meta data cache to isolate query as a subset of search space.

3.1 Meta data cache mechanism

The meta data cache stores guide information to reduce the exhaustive searches of many hierarchical name services, like X.500, as a result of reducing the search space. The decision of search space, in case a new query happens, derives direct DSA access after monitoring the similarity with the queries that happened before. To do these things, the queries that happened before are reorganized as a meta data tree and stored into the meta data cache. The storage structure of the meta data tree is shown as in (Figure 1) and its organization holds a tree form as shown in (Figure 2).

level	data	ptr_CNL	a_p	next
depth of entry	entry name	pointer of child	access point	pointer of sibling

(Figure 1) Storage structure of meta data cache



(Figure 2) Structure of meta data tree

Each node on the meta data tree includes the entry level, an entry name, a child node pointer that points subentry, a DSA access point that includes an entry, and a sibling node pointer that points to an entry on the same level.

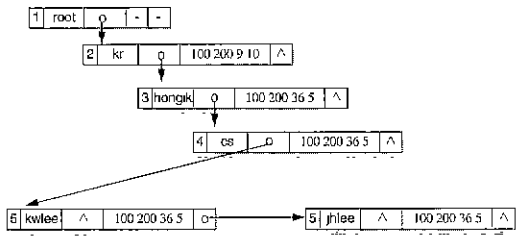
3.1.1 Adding an entry into meta data tree

After getting the result of a query which happened, the query and its result are stored into the data cache, and meta data about the query should be stored in the meta data tree for the reuse in near future. The addition process to the meta data cache using the query below can be performed as follows.

- **query 1** : select * from people where CommonName = 'kwlee' and Country = 'kr'
- **query 2** : select * from people where CommonName

= 'jhlee' and Organization = 'hongik' and Country = 'kr'

With the state that the result of **query 1** was already formed in meta data tree, if the result of the **query 2** is arrived, it should be added to the meta data tree as a new node through the add algorithm. In case of addition, it is added to the meta data tree as a new node through the mapping process with an existing meta data tree. (Figure 3) shows the result of reorganization after adding the information of **query 2** The adding algorithm of the meta data tree is shown in (Figure 4).



(Figure 3) 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
```

(Figure 4) Adding algorithm

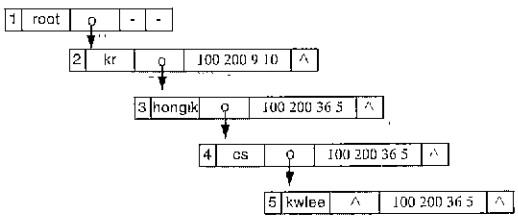
3.1 2 Searching meta data tree

The process of the decision of search space through the search process of the meta data tree stored in the meta data cache using queries below is as follows.

- **query 3** : select * from people where CommonName = 'hschung' and Organization = 'hongik' and Country = 'kr'

The data cache (application data query cache or system data query cache) is empty state because it is an initial state when the **query 1** happens

So, the response for a query will be acquired through the search function of X.500 after searching the numerous DSA's. The result that is acquired by the **query**, first, is shown to the user who requests the query And after deciding whether it is the application data query cache, the response of the query is stored into the application data cache. And in case of the system data query, the query is stored into the system data query cache. In the meta data cache, the information about the **query 1** is reorganized as the meta data tree as shown in (Figure 5). If the **query 3** is arrived, on these states, it is checked whether there exists the same query in both the application data query cache and system data query cache. In the above case, a cache miss happens, so the search space should be decided using the meta data tree stored in the meta data cache. It is clear that the search space of the **query 3** is **"Organization = hongik"** through the search algorithm of the meta data tree in (Figure 6). So, the **query 3** can access the DSA that includes **"hongik"** entry directly with the access point of **"100.200.36.5"** of the **"hongik"** stored in the meta data cache. The search algorithm of the meta data tree is shown in (Figure 6).



(Figure 5) Search example of meta data

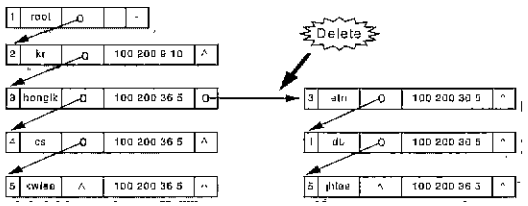
```
SEARCH_DIT_SNODE(query)
/* structure { c, o. ou, p } query  /
if DIT_SNODE.ptr_CNL = NULL then return
else if SEARCH_LIST(query c) = FALSE then return
 else DIT_SNODE ' = CURRENT_LIST.CHILD
  if SEARCH_LIST(query o) = FALSE then
   DATA_ACCESS(DIT_SNODE cacheAddr)
    else DIT_SNODE  = CURRENT_LIST.CHILD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

(Figure 6) Searching algorithm

3.1 3 Deleting a node from meta data tree

Deleted node in the meta data tree is a node of query information about entry which should be removed from the data cache When has a data replacement by full of the data cache, deleting a node from the mete data tree is occurred.

If the meta data tree is stored in the meta data cache as shown in (Figure 7) and the query of **"select * from people where CommonName = 'jhlee' and Organization = 'etri' and Country = 'kr'"** is removed from the data cache, then the nodes must be deleted from the meta data tree. The deleting algorithm of the meta data tree is shown in (Figure 8)



(Figure 7) Example of deleting meta data tree

```
DELETE_DIT_SNODE(entry, LEVEL,DELETED)
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
```

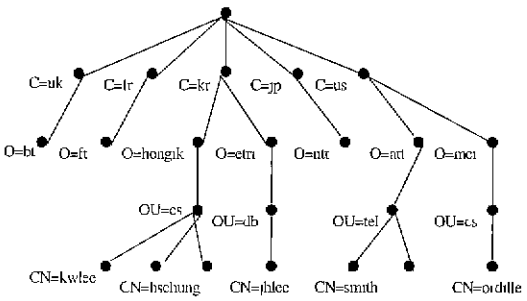
(Figure 8) Deleting algorithm

4. Performance measurements

In this section, we compare the case of using the meta data cache with the case of not using it, and evaluate both cases.

4.1 Comparison of using meta data cache

Assume that the DSA in the distributed directory environment is located at the level of **"OrganizationUnit(ou)"** in the (Figure 9), and that number of **"Country(c)"** is M, **"Organization(o)"** is N, and **"OrganizationUnit(ou)"** is L



(Figure 9) Process environment of example query

The average access frequency to DSA for each query is shown in <Table 1>. When the number of **"Country(c)"**, **"Organization(o)"**, and **"OrganizationUnit(ou)"** changes by step respectively as shown in <Table 2> And in both cases of using the meta data and not using it, the comparison of the average access frequency to DSA for each query is shown in (Figure 10) and (Figure 11). Especially, in the case of **query 6**, **query 7** and **query 8**, the DSA access frequency for searching object using the meta data is more reduced than that of using the X.500. And (Figure 12) is comparison of performance result in proposed model and X.500.

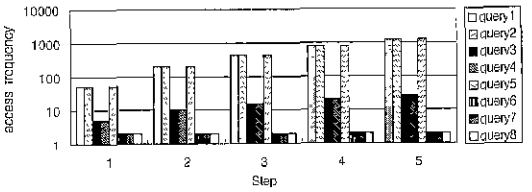
〈Table 1〉 Average access frequency to DSA

Query	Using the meta data	Not using the meta data
query1	(M+N*L)/2	(M+N*L)/2
query2	L/2	(M*N*L)/2
query3	(N-L)/2	(M-N*L)/2
query4	L/2	(M-N*L)/2
query5	1	0
query6	(M+N*L)/2	(M+N*L)/2
query7	(N*L)/2	(M+N*L)/2
query8	L/2	(M-N+L)/2

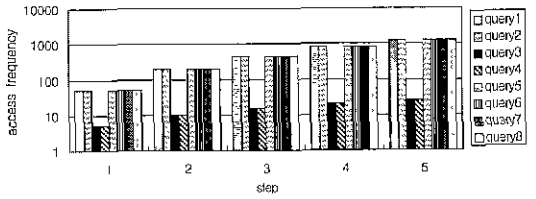
- **query 1 : select * from people where cn = "kwlee" and c = "kr";**
- **query 2 : select * from people where cn = "hschung" and c = "kr";**

〈Table 2〉 Change number of the DSAs by step

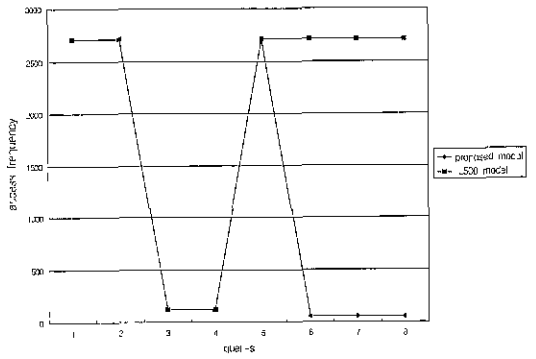
Entry	Steps				
	1	2	3	4	5
Country	1	1	1	1	1
Organization	10	20	30	40	50
OrganizationUnit	10	20	30	40	50
Total	100	400	900	1600	2500



(Figure 10) Performance result in proposed model



(Figure 11) Performance result in X.500



(Figure 12) Comparison of performance result in proposed model and X.500

- query 3 : select * from people where cn = "hschung" and o = "hongik" and c = "kr";
- query 4 : select * from people where cn = "kwlee" and o = "hongik" and c = "kr";
- query 5 : select * from people where cn = "kwlee" and o = "etri" and c = "kr";
- query 6 : select * from people where cn = "kwlee" and ou = "db" and c = "kr";
- query 7 : select * from people where cn = "hschung" and ou = "cs" and c = "kr";
- query 8 : select * from people where cn = "psshin" and ou = "cs" and c = "kr";

5. 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 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, system data and meta data. Second, we designed the structure of the cache system and the storage structure of the classified cache information. Third, we propose an operational algorithm of the meta data cache which stores the meta data tree organized by previous queries to speed up the query processing by reducing the search space of query. Finally, we show that the number of access frequency to DSA in the proposed model using the meta data is more reduced than the object searching technique of X.500. The overhead of our proposed method is a cost of cache processing in a local site. But we didn't consider that time because the costs are little than communication costs

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 ICDCS, 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] Yixiu Huang, Robert H. Sloan, Ouri Wolfson, "Divergence Caching in Client-Server Architectures." Proceedings of the third ICPDIS Austin, TX, Sept. 1994.

[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] J. H Lee, "A Design of Object-Active-Knowledge (OAK) Directory Database Model for Telecommunication," Ph.D. Thesis, Hongik Univ. 1996. 8

[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, Wisconsin Univ., Nov. 1993.

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

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, 객체지향 데이터베이스, 분산 시스템

고 진 광

e-mail : kjg@sunchon.ac.kr

1982년 홍익대학교 전자계산학과 졸업(이학사)

1984년 홍익대학교 전자계산학과 졸업(공석사)

1997년 홍익대학교 전자계산학과 졸업(이학박사)

1984년~1988년 송원전문대학 전자계산과 전임강사

1988년~현재 순천대학교 컴퓨터과학과 정교수

1993년~1994년 홍익대학교 컴퓨터공학과 국내교류교수

1997년~1998년 오리건주립대학교 컴퓨터공학과 방문교수

관심분야 : 데이터베이스, 트랜잭션 관리, 정보통신, CALS/EC