December 2001

Large Object Caching for Distributed Multimedia Information Systems

Hyunchul Kang
Vienna University of Technology

Jong-Min Lee
Chung-Ang University

LARGE OBJECT CACHING FOR DISTRIBUTED MULTIMEDIA INFORMATION SYSTEMS

Hyunchul Kang
Chung-Ang University
hckang@rose.cse.cau.ac.kr

Jong Min Lee[1]
Chung-Ang University
lons@rose.cse.cau.ac.kr

Abstract

The large object(LOB) such as text, image, audio, video, etc. has become one of the typically encountered data type in today's internet/intranet-based distributed multimedia information system for business and other applications, and thus, needs to be efficiently retrieved from the remote databases. As such, its underlying database management system(DBMS) is required to efficiently support LOBs through its application programming interface(API). In this paper, we propose LOB caching through the SQL Call Level Interface, a widely employed API for database access, whereby the frequently accessed LOBs could be retrieved locally. The proposed extension was implemented on a DBMS, and the performance assessment of LOB retrieval with and without caching is reported.

Introduction

One of the technical challenges to the modern information system in today's internet/intranet-based computing environment is that it is required to efficiently retrieve *large objects(LOBs)* such as text, image, audio, video, etc., of which multimedia data is composed, from the remote database servers over the communication network. Although today's communication networks are getting faster, retrieving a LOB from the remote database still takes long primarily due to its huge volume. No matter how fast the network may be, at least the propagation delay should be tolerated. Besides, the network congestion under the usual daytime heavy loads may be unavoidable. As such, the underlying database management system(DBMS) for a *distributed multimedia information system* is required to efficiently support LOBs through its application programming interface(API).

The *SQL Call Level* Interface*(CLI)* is a widely used API for database access (Venkatrao and Pizzo 1995), which fits the dynamic, open, and distributed information system environment. For example, the SQL CLI is the appropriate choice in developing an application program(AP) such as the front-end of an information system running on a PC client accessing the back-end remote database servers across a communication network.

The SQL CLI is a set of functions which get an AP connected to databases, request the execution of the SQL commands, receive their results, and so on. The SQL CLI functions called by an AP are executed through the *CLI library*, a middleware working between the AP and the target DBMS. Each DBMS that supports the SQL CLI needs to provide its own CLI library, which is implemented as a dynamic library to be linked to the APs at run time (see Fig. 1(a)). If the CLI library for a particular DBMS conforms to some standard specification of the SQL CLI (e.g., X/Open SQL CLI (X/Open 1992), ISO-ANSI SQL CLI (Melton 1993), ODBC interface (Microsoft 1997), and JDBC interface (Fisher et al. 1999)), any AP conforming to that standard is binary interoperable with that DBMS.

In this paper, we investigate an extension of the SQL CLI to make it a more efficient API in retrieving LOBs from the remote databases, enabling development of high performance distributed multimedia information systems. We propose that the SQL CLI provide the new functions for *LOB caching* capability whereby the frequently accessed LOBs can stage in the local storage for fast retrieval later on. As shown in Fig. 1(b), if the conventional CLI library is extended with a *LOB cache manager* for storing

[1]Current address: VOIN Technology Co., Ltd., Seoul, 150-032, Korea.

LOBs in the local cache, the AP could retrieve LOBs from the cache instead of from the remote databases over the communication network. Observing that LOBs such as images and videos in multimedia information systems are mostly read-only and rarely change, LOB caching is a promising approach in distributed environment.

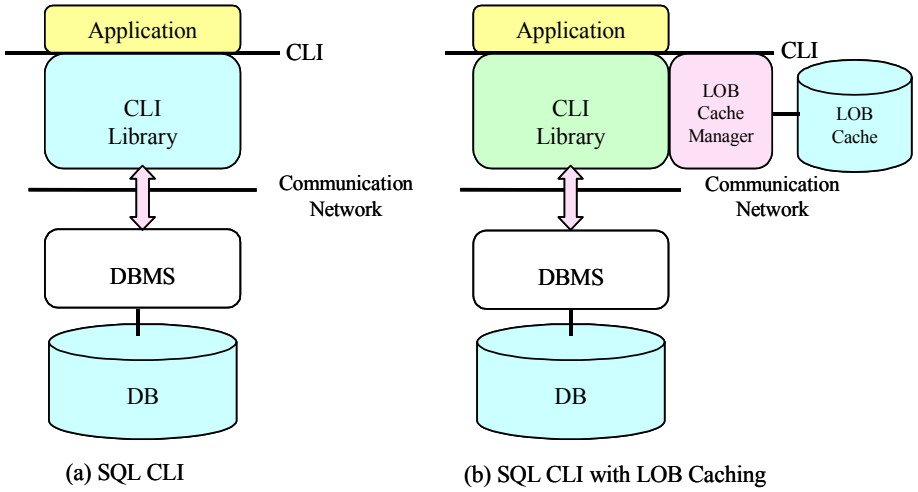

Figure 1. SQL CLI and LOB Caching

The technical issues with LOB caching through the SQL CLI include *cache consistency*, *cache replacement*, *management of cache information*, and the *functionalities to be added to the traditional SQL CLI*. Section 2 describes how these issues are dealt with in our extension. The proposed extension has been implemented on a DBMS which supports the SQL CLI specified by X/Open, and we have conducted some experiments on this implemented system for performance assessment of LOB retrieval with and without caching. Section 3 reports the experimental results. Finally, some concluding remarks are in Section 4.

Technical Issues with LOB Caching through SQL CLI

Maintaining Consistency of the Cached LOBs

Consistency maintenance is a key requirement to support LOB caching through the SQL CLI. Although the problem of consistency maintenance with the traditional page caching or with query caching was extensively dealt with in the client-server DBMS context (Franklin 1996)(Delis and Roussopoulos 1993)(Park and Kang 1998), that with LOB caching was rarely investigated.

We adopt the *check-on-access* policy where validity of a LOB in the cache is checked with the database server when the LOB is requested. If the requested LOB in the cache turns out to be valid, it is retrieved from the cache, and if it does not, it is retrieved from the remote database server and cached. In this process, the LOB identifier(ID) and its timestamp(TS), which indicates the time when the database server last updated the LOB, are referred to. As such, when a LOB is stored at the database server, it is assigned a unique ID, which is used for checking whether or not a certain LOB exists in the cache, and when a LOB is cached, its ID and TS are stored in the cache as well.

The process of caching and retrieving a LOB while preserving cache consistency is shown in Fig. 2. When the AP requests a LOB through the SQL CLI with a query (for example, retrieve the chart depicting the sales last month)(Fig. 2-1) the CLI library first retrieves the ID and the TS of the corresponding LOB from the database server(Fig. 2-2 and 2-3). And then, with the returned ID, it is checked whether or not the requested LOB is in the cache(Fig. 2-4). If it is in the cache, its cached TS is compared to the TS just retrieved from the database server to see if the cached version of the LOB is still up-to-date. After all, there could be three cases:

1. The requested LOB is in the cache, and it is up-to-date.
2. The requested LOB is in the cache, and it is outdated.
3. The requested LOB is *not* in the cache.

In the first case, the requested LOB is retrieved from the local cache(Fig. 2-5), avoiding further communication with the database server which might take very long. In the other cases, the requested LOB is retrieved from the database server over the communication network and cached(Fig. 2-6 through 2-8). In the second case, the original LOB in the cache is purged and replaced with the up-to-date version retrieved. In all cases, the CLI library returns the retrieved LOB to the AP(Fig. 2-9), and keeps the up-to-date version of the LOB in the cache.

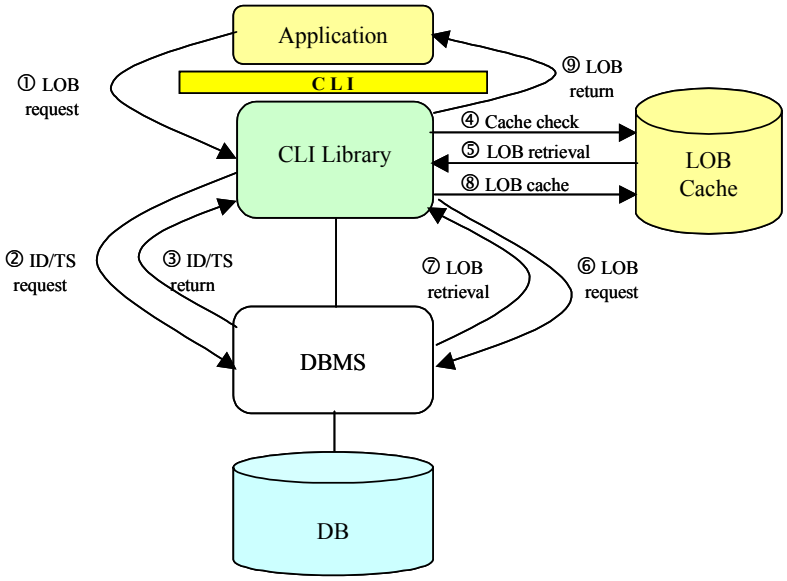

Figure 2. Process of Retrieving and Caching a LOB while Preserving Cache Consistency

LOB Cache Replacement Algorithm

The conventional DBMS buffer caches a portion of the database in units of disk pages. Much research has been done on the page replacement algorithm such as LRU (Effelsberg and Haerder 1984). A replacement algorithm is also necessary with LOB caching, and it should be somewhat more complicated one compared to those for page replacement because the LOBs are stored in various sizes usually much lager than the size of a page. For example, the algorithm such as LRU-SIZE, LRU-MIN, and LRU-Threshold, which are all LRU-based, are for such purposes (Abrams et al. 1995). Any of these algorithms is applicable to our extension, and LRU-MIN is the choice for our implementation of the extended SQL CLI.

LRU-MIN works as follows. It picks the victim using LRU among the LOBs whose sizes are larger than the size of the LOB that is to be newly stored in the cache. If there is no such LOB, the size of the new LOB is divided by two, and the victim selection is done using LRU against those LOBs whose sizes are larger than the divided size. And if there is not yet enough space after the replacement of the victim(s), the divided LOB size is divided again by two, and the same process is repeated until there is enough space for the new LOB.

Efficient Management of LOB Cache Information

In order to efficiently support LOB caching, it is necessary to efficiently maintain and manage the LOB cache information. There are two types of LOB cache information. One is the general information on all of the cached LOBs, and the other is the individual information on each of the cached LOBs. The general information includes the information on the *LOB cache area* which means the storage area dedicated for the cached LOBs, the statistical information on the cached LOBs, and the access path information for cache retrieval. The individual information includes the LOB identifier, the timestamp, the last time of reference, and the size. The LOB cache information needs to be stored along with the cached LOBs in the cache to be efficiently accessed, and thus, some efficient data structures are required to store it. In our implementation, we devised a set of hashing-based data structures similar to those employed for the lock manager implementation for DBMSs described in (Gray and Reuter 1993, Chap. 8).

New SQL CLI Functions for LOB Caching

The functions that need to be added to the conventional SQL CLI for LOB caching can be classified into four categories as summarized in Table 1. The first category is for allocating and freeing a *LOB cache handle*, which is a pointer to the data structures storing the LOB cache information. Its role is similar to those of the environment handle, the connection handle, and the statement handle used in calling the SQL CLI functions (X/Open 1992). That is, the AP is to call the functions related to LOB caching with the allocated LOB cache handle as a parameter. The function named *AllocLOBCacheHandle()* allocates and initializes the memory for the data structures to store the LOB cache information, and also sets a pointer in the LOB cache handle

to point to these data structures. *FreeLOBCacheHandle()* deallocates the data structures from the memory and sets the pointer in the LOB cache handle as NULL.

Table 1. New SQL CLI Functions for LOB Caching

Category	CLI Function	Description
Allocating and Freeing the LOB Cache Handle	AllocLOBCacheHandle()	allocates a LOB cache handle
	FreeLOBCacheHandle()	frees a LOB cache handle
Allocating and Freeing the LOB Cache Area	AllocLOBCache Area()	allocates a new LOB cache area
	FreeLOBCacheArea()	deletes an existing LOB cache area
Retrieving/Caching the LOBs and Deleting the Cached LOBs	GetnCacheLOB()	retrieves and caches a LOB
	PurgeLOB()	deletes a LOB from the cache
Retrieving and Modifying the LOB Cache Information	GetLOBCacheInfo()	retrieves the LOB cache information
	SetLOBCacheInfo()	modifies the LOB cache information

The functions for allocating and freeing the LOB cache area are also necessary. *AllocLOBCacheArea()* executes the process of setting up a directory in the storage space where the LOB cache area is installed, and updates the LOB cache handle accordingly. *FreeLOBCacheArea()* is to delete the directory. Several LOB cache areas could be maintained separately at the same time for more efficient LOB caching. For example, the LOBs that are more frequently accessed can be stored in the cache area installed on a faster storage device while less frequently accessed LOBs are stored in another cache area installed on a slower device.

The third category of functions is the core in our extension: a function for retrieving and caching a LOB and a function for purging a LOB from the cache. In the SQL CLI specification by X/Open, the function named GetCol() is to be called for LOB retrieval. In order to support LOB caching, GetCol() needs to be extended to *GetnCacheLOB()* so that it can not just perform the original functionality of GetCol() but also check if the requested LOB is in the cache, and if so, check if it is still valid, and retrieve it from the cache instead of from the database server if it turns out to be valid. Meanwhile, since the LOB cache area is with space limitation, some of the cached LOBs should be purged when not hot any more. For this, we need *PurgeLOB()* for deleting certain designated LOB or the LOBs that have been last accessed long ago.

Finally, the functions are required for retrieving and modifying the LOB cache information. *GetLOBCacheInfo()* is to retrieve the general and/or the individual information of the cached LOBs, whereas *SetLOBCacheInfo()* is to modify those information. These functions are useful for efficient management of the LOB cache areas.

Performance Assessment of LOB Caching

The proposed extension of the SQL CLI was implemented on a DBMS. It was originally equipped with a CLI library conforming to the SQL CLI specification by X/Open, and that was extended to incorporate our new SQL CLI functions for LOB caching. The extended CLI library was implemented in Microsoft Visual C++ 6.0 on Windows NT 4.0, and communicates with the remote database server through the Ethernet.

We have conducted some experiments with our implementation for performance assessment of LOB caching, comparing the LOB retrieval time with GetnCacheLOB() to that with the conventional GetCol() specified by X/Open. The retrieval time has been measured with respect to the various LOB sizes and cache hit ratios. The LOB sizes employed are 1MB, 5MB, 10MB, 20MB, 30MB, 40MB, and 50MB, and the cache hit ratios experimented are 0%, 20%, 50%, 80%, and 100%. The unit of the LOB retrieval time is the second, and the retrieval time of each LOB was measured 20 times and averaged.

Figure 3 shows the results. With GetnCacheLOB(), the best performance is achieved when the cache hit ratio is 100%, whereas the worst is observed when the cache hit ratio is 0%, as expected. Note that the cache hit here counts only for those cases where the requested LOB is in the cache *and* it turns out to be still valid. In other words, if the requested LOB is found in the cache but it turns out to be stale, and thus, it is to be retrieved from the remote database server, it is considered as a cache miss.

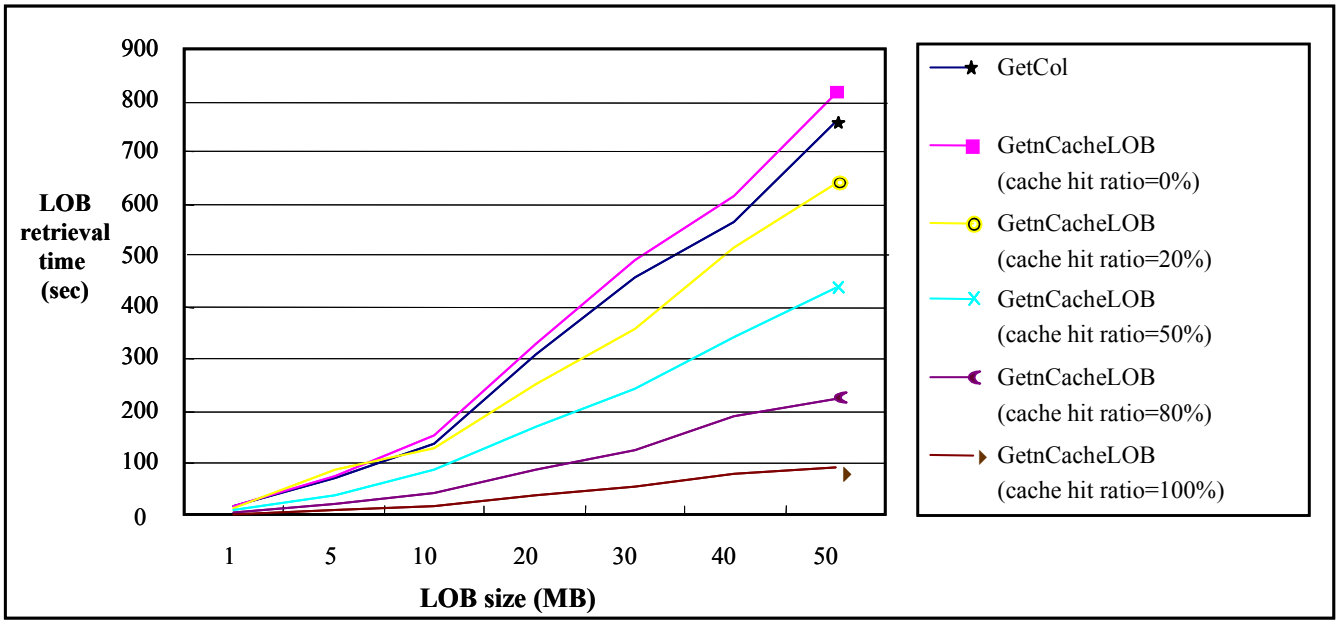

Figure 3. Performance Comparison between GetnCacheLOB() and GetCol()

When the cache hit ratio was 0%, LOB retrieval with GetCol() slightly outperformed GetnCacheLOB(). This is because GetnCacheLOB() takes some time for retrieving the identifier and the timestamp of the requested LOB first for the purpose of searching the cache for the requested LOB just to find out it does not exist there, and eventually some additional I/O time is spent for storing the requested LOB retrieved from the database server in the cache. However, when the cache hit ratio was 100%, GetnCacheLOB() considerably outperformed GetCol(). The performance improvement with cache hit ratio of 100% is much larger than the performance penalty with cache hit ratio of 0%. For example, when a cache miss occurs in retrieving a LOB of 50MB, GetnCacheLOB() suffered from about 7% performance degradation compared to GetCol(). However, with 100% cache hit, GetnCacheLOB() gained 88% performance improvement over GetCol().

Now what if the cache hit ratio is not so perfect as 100%? Could the performance gain through caching with cache hit compensate for the performance loss with cache miss? Fig. 3 shows that the LOB retrieval time increases as the cache hit ratio decreases. However, GetnCacheLOB() does not take so much time as GetCol(). For example, when the cache hit ratio was just 20%, the time it took to retrieve a LOB of 50MB with GetnCacheLOB() showed about 21% improvement compared to that with GetCol(). It implies that if only 20% of the desired LOBs could be retrieved from the local cache instead of the remote database server, resorting to the LOB caching capabilities in the extended SQL CLI is profitable. The improvement achieved here is still much bigger than the 7% overhead that would incur if the cache hit ratio were 0%.

Concluding Remarks

LOB caching through the database API proposed in this paper can be an important vehicle in realizing high performance distributed multimedia information systems for numerous applications such as geographical information systems, cyber education, internet entertainment, and E-catalogues for cyber shopping malls to name just a few. LOBs such as images and videos in those applications are mostly read-only and rarely change, and thus, their caching could be quite efficient.

Towards a full-fledged multimedia information system efficiently dealing with LOBs, some further work needs to be done on extending the conventional database APIs. The technical issues deserving exploration in this regard include the new features for *LOB manipulation* and *definition,* which are more powerful and flexible compared to those provided in the conventional database APIs.

References

Abrams, M., Standridge, C., Abdulla, G., Williams, S., and Fox, E., "Caching Proxies: Limitations and Potentials," *Proc. of the Fourth Int'l Conf. on the World Wide Web*, December 1995.

Delis, A. and Roussopoulos, N., "Performance Comparison of Three Modern DBMS Architectures," *IEEE Trans. on Software Engineering* 19(2), Feb. 1993, pp. 120-138.

Effelsberg, W. and Haerder, T., "Principles of Database Buffer Management," *ACM Trans. on database Syst.* 9(4), Dec. 1984, pp. 560-595.

Fisher, M., Cattell, R., Hamilton, G., White, S, and Hapner, M., *JDBC API Tutorial and Reference*, Second Edition, Addison Wesley, Jun. 1999.

Franklin, M., *Client Data Caching: A Foundation for High Performance Object Database Systems*, Kluwer Academic Publishers, 1996.

Gray, J. and Reuter, A., "Transaction Processing: Concepts and Techniques," Morgan Kaufmann, 1993.

Melton, J. (Ed.), "ISO-ANSI (Working Draft) SQL Call-Level-Interface(CLI)," *ISO DBL MUN-005/ANSI X3H2-93-360*, Aug., 1993.

Microsoft Corp., *ODBC 3.0 Programmer's Reference and SDK Guide*, Microsoft Press, 1997.

Park, K. and Kang, H., "A Client group-Server DBMS Architecture and Inter-Client Communication Caching Schemes in the WAN Environment," *Proc. Int'l Conf. On Cooperative Information Systems*,1998, pp. 198-207.

Venkatrao, M. and Pizzo, M., "SQL/CLI - A New Binding Style for SQL," *ACM SIGMOD Record* (24:4), Dec. 1995, pp. 72-77.

X/Open Company Ltd., *Data Management: SQL Call Level Interface (CLI), Snapshot*, Sep. 1992.