

# The CACTUS Object Server: Design and Implementation Issues\*

I-Heng Meng      Wei-Pang Yang

*Institute of Computer and Information Science*

*National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.*

[meng@dbsum1.cis.nctu.edu.tw](mailto:meng@dbsum1.cis.nctu.edu.tw)      [wpyang@cis.nctu.edu.tw](mailto:wpyang@cis.nctu.edu.tw)

Swu-Mei Lin, Mu-Ann Chen, Shih-Kung  
Chen, Jian-Cheng Dai

*Institute for Information Industry, Taiwan, R.O.C.*

## Abstract

*CACTUS is a compact and reliable object storage system based on peer-to-peer architecture with distributed transaction processing capability. The kernel is implemented with the multi-thread mechanisms to provide a high performance multi-transaction environment. All sites, installed with CACTUS, of the networked workstations are on an equal footing. The system can handle local and remote transactions including accessing remote objects or processing remote accesses. In this paper, we give an architectural overview of CACTUS and discuss some implementation issues. It mentions the special contributions such as transient large objects, the coarser lock granularity and its associated protocol. A cost model is developed to demonstrate that the new scheme yields performances which are superior in most common cases and inferior under certain situations.*

## 1. Introduction

CACTUS, a Collaborated Autonomous Chiao-Tung Universal Server [Yang96, Yang97], is an object database system researched and developed by the Database System Lab of National Chiao-Tung University, Taiwan, R.O.C. The inner design of this system is an object server and a

more flexible peer-to-peer architecture. The system is a compact and quick kernel with the ability to coordinate multiple transactions in multi-users and multi-tasks fashion. In order to increase the system throughput, the kernel uses multi-thread mechanism to speed the response time by parallel executing threads. Meanwhile, based on the basic object server architecture for getting a higher hit ratio on cache, we enhance the ability to transform a group of objects between client and server. A new lock granularity is also introduced in our design in order to reduce the communication overhead while objects are accessed. The kernel also provides essential management functions such as object, index, very-large-object and distributed transaction management.

CACTUS, based on peer-to-peer architecture, possesses the distributed transaction processing ability. The overall architecture is shown as Figure 1.1. All sites, installed with CACTUS, of the networked workstations are on an equal footing and can process both local site transactions as well as serving remote site transaction requests. While accessing objects, users should clearly know the site where objects reside and directly specify the *SiteId* parameter of the corresponding APIs upon calling. System can handle two types of transactions, local transactions and remote transactions, under the control of transaction manager.

\* This research was sponsored by MOEA and supported in part by Institute for Information Industry, Taiwan, R.O.C.

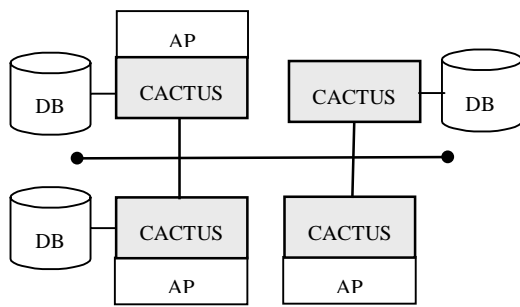


Fig. 1.1 The overall architecture of CACTUS.

### 1.1. Architecture

CACTUS runs on the Windows<sup>®</sup> 95 platform. All of workstations installed with CACTUS are linked together to form a distributed database environment. The system architecture of CACTUS is shown in Figure 1.2 to represent the design of layer structure.

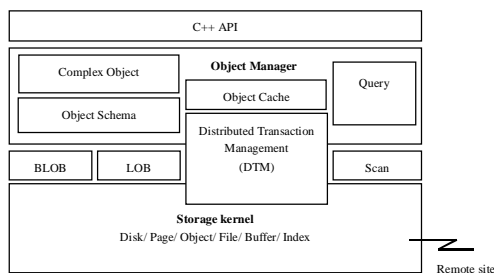


Fig. 1.2 The system architecture of CACTUS.

In CACTUS, there are two types of transactions which are local transactions and remote transactions for each site. The former represents local executing transactions which may access local or remote objects. The latter acts as transactions which are executing at remote sites. User transactions can access any objects across the networked sites as long as these sites are installed with CACTUS systems.

User transactions running on the local site are called *main transactions*, while those transactions running on the remote sites are called *subtransactions*. Each transaction is processed by a main transaction via the main socket identifier after the transaction begins. When the

transaction requests a remote data, the request will be passed to the specified remote site right away. If the user transaction requests to commit (or abort), the main transaction accompanies with all remote subtransactions need to commit (or abort) all of their executions. The two-phase commit (2PC) protocol is implemented in CACTUS to insure the consistency.

The remainder of the article is organized as follows: Section 2 presents some implementation issues of CACTUS in detail. Section 3 analyzes the performance improvement of group object server that compared to pure object server, and observes some effects by tuning some parameters. Section 4 concludes this paper and outlines some future researches.

## 2. Some implementation issues

CACTUS adopts ARIES [Moha92] as the log and recovery protocol and is fully implemented. The system provides the multi-thread processing capability in order to facilitate the multi-transaction environment. Each client transaction establishes a dedicated connection with server, that is, each client has a fixed and unique socket identification associated with the server.

### 2.1. Object storage

In CACTUS, objects are classified into regular objects and large objects. The size of regular object is bounded within the fixed page size - 4Kbytes in our current design. Yet the size of large objects is greatly beyond the limit. The system provides a single set of APIs and users do not know completely the inner manipulations between the two types of objects, they are automatically transformed into each other if it is necessary after judging the size of the object.

The transformation is triggered if the size of regular object is larger than the fixed page size (4Kbytes). The

regular object consists of a header and an object body. When the transformation begins, the system uses the object body of the regular object to store the related information for the large object. First, the system will calculate the number of page frames needed to store the large object, then places the counted number on the first two bytes, the following bytes denoted by  $\rho$  are a set of page pointers to the page frames that store the content of large object. The rest *unused space* of object body as shown in the shadowed area will be collected by the page manager and adjust the related page control information in the page header. The memory structure of a regular object before transformation is plotted as in Figure 2.1(a), and a large object after transformation is shown as in Figure 2.1(b).

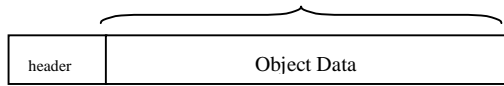


Fig. 2.1(a) The structure of a regular object.

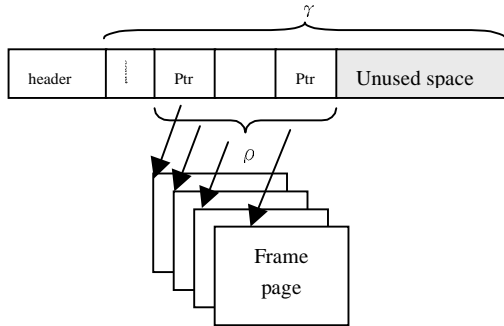


Fig. 2.1(b) The structure of a large object.

## 2.2. Coarser concurrency control granularity

Although objects are the principal granularity of data transfer in object server, concurrency control can be performed at a coarser granularity. We introduce a new lock granularity, called **in-page-class lock**, which is coarser than an object lock and finer than a file lock in our system. The new granularity locks all objects that with the same class in a page. The group is constituted by set of objects that belong to the same class and it is called **in-**

**page-class group**. The lock granularity hierarchy of CACTUS is illustrated as Figure 2.2.

The lock compatibility matrix shown as in Table 2.1 is applied. The Read and Write modes have the same semantics and meaning as the conventional 2PL. Two additional lock modes, **cached lock** and **scanned lock**, are also introduced.

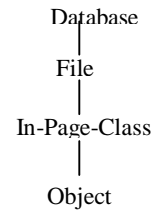


Fig. 2.2 The granularity hierarchy of CACTUS.

Lock already set	Lock to be set		
	Scanned	Read	Write
Cached	✓	✓	✓
Scanned	✓	✓	✓
Read	Unchanged	✓	✗
Write	Unchanged	✗	✗

Table. 2.1 The lock compatibility matrix of CACTUS.

Each transaction starts with the coarser in-page-class granularity and checks the lock compatibility of the in-page-class group that contains the requesting object. In the absence of data conflicts about the coarser granularity, the system operates in in-page-class locks. Otherwise, the in-page-class lock is de-escalated to finer object locks. The system considers those objects in the in-page-class group as a logical unit which is the basic transferred unit between client and server. While checking lock conflicts, one of the four situations as follow may arise:

### Case 1: No In-Page-Class Lock Conflict

- (1) Multiple client sites may read different groups from different pages or the same group of objects in a page with the read lock modes.

- (2) Only one client can lock the group of objects with write lock mode. If the other transactions reference the same group, the lock de-escalation will be initiated by the writer and performed as the case 2.
- (3) The requesting object in the group is set with the lock mode given by the user, the other objects in the group are set to the cached locks.

#### **Case 2: In-Page-Class Lock Conflict**

- (1) The owners of in-page-class locks, who hold the copy of the group, are asked to de-escalate to object level locks. All the in-page-class group owners return the list of object locks that only consist of read locks and write locks but it without needing the cached locks and scanned locks.
- (2) The server merges all object locks come from all sites that had accessed the group of objects before, and properly keeps the location information of these objects and sets the lock table with their lock modes. At the same time, the server also adjusts the flag in in-page-class lock table to indicate the done de-escalation action of that group. The flag tells the subsequent reader directly get the object in object level and bypasses the unnecessary group level operations.
- (3) After de-escalation is done, the server will check object level conflicts as the next two cases.

#### **Case 3: No Object Lock Conflict**

- (1) If the lock request is permitted and the object is hit in the cache, the transaction will get the object directly and set the proper lock information at client site. However, if the lock request is granted but the object is miss at client cache, the transaction will acquire the lock at server. Group of objects are retrieved and transmitted from the server except for these objects which are accessed by other clients with write lock modes are marked invalidated.

- (2) When a transaction need to update an object and the lock is granted, the object's owners known from the server location table will be notified and invalidate the object in their inter-transaction cache. If the object lock is incompatible, then go to case 4.

#### **Case 4: Object Conflict**

- (1) When a client requires to update an object, the system has to promote the read lock acquired earlier to an object-level write lock. If the requested object is also read locked by other clients, the requestor will be blocked until the lock is released.
- (2) The requesting client requires to read an object but it is locked by the other client with write lock mode, then the requestor is also blocked.

We introduce a lock mode called **cached lock**. Objects in the group, which are not referenced and locked by any transactions, are given the cached lock modes. The transaction can read objects which are in cached lock mode instantly without any checking at the client, and the cached locks will be promoted to read locks after the read operations complete. Nevertheless, under the situation of updating objects that are in read locks at client, the client has to send the request to the server in order to get the write lock. The requested objects may still in cached lock modes or in read lock modes at other client's cache, but the server still keep the object in cached lock mode. If the requested object is in cached lock mode at the server, then it callbacks all locks holders and sets the write lock of the requestor at the server as long as all callbacks are returned compatible. If the lock holder receives a callback request and the lock is in read lock mode, the write lock request is incompatible and return fail. This results the write lock request fails and waits. When the client receives a callback request and check compatible, it purges the cached lock and the cached object from the client.

In order to achieve this goal, the server needs to keep

tracking the locations of cached objects and groups in object-level and in-page-class granularity. We introduce another lock mode called the **scanned lock**. The lock mode of cached objects will be changed from cached lock to scanned lock after performing scan operations on these objects. Therefore, objects still stay in cached lock mode will be replaced by cache replacement strategy while cache space is ready to full.

### 3. Performance analysis

We analyze the performance of the proposed object server architecture. Some metrics have to be evaluated in our experiment.

- (1) To evaluate the performance gain results from the new scheme.
- (2) To examine what degree the parameters of the new scheme affect the whole system performance.

To achieve this measurement we use scan operations for the test. A cost model will be derived so as to evaluate the access time of new scheme as well as considering the effect of concurrency control. A metric called *Average Object Access Time (AOAT)* is defined to find out the cost of an object access in terms of elapse time during the execution of a transaction. To simplify the discussions, some assumptions are given in our evaluation:

- (1) There are two levels of memory hierarchy in CACTUS, which are object cache at client and page buffers at server.
- (2) The following analyses are all based on the scan read operations. All object scans do not via any index, they are pure object file accesses.
- (3) For simplicity purpose of the experiment, the concurrency control and the deadlock detection both are ignored for the cost model.

### 3.1. Cost model

Before we state the cost model of the above two cases, we take a look at the buffer accesses in advance which are related to the cache miss rate of client cache. When objects are not accessed via an index it can be assumed that the objects are randomly accessed. The cost function given by Yao [Yao77], which optimally estimates the number of unique pages to be accessed. Given  $n$  objects grouped into  $m$  pages ( $1 < m \leq n$ ), each contains  $n/m$  objects. If  $k$  records ( $k \leq n - \frac{n}{m}$ ) are randomly selected from the  $n$  records, the expected number of pages hit is given by the following:

The scan operation will get all objects  $O_{c_i}$  with class  $C_i$

$$Y(k, n, m) = m \times \left( 1 - \frac{\binom{n - n/m}{k}}{\binom{n}{k}} \right) \quad (3.1)$$

and no repetition is allowed. Its behavior is pretty much like the Brute Force access method. The number of in-page-class groups accessed for the scan is  $S_{c_i}$  among all of pages  $P_{c_i}$ . In practice, all pages in the database do not necessary contain the objects of class  $C_i$ . So  $P_{c_i}$  is not necessary equal to the total page  $P_{c_i}^*$  of database contained.

$$S_{c_i} = Y(O_{c_i}, O_{c_i}, P_{c_i}, 1) = Y(O_{c_i}, O_{c_i}, P_{c_i}) \quad (3.2)$$

According to the corollary of Yao's [Yao77], we may conclude that  $S_{c_i} = P_{c_i}$ . If enough cache space is allocated, there are no cache misses at all. Otherwise, there are  $S_{c_i}$  objects requesting  $S_{c_i}$  pages (which are called *first reference misses*) and the other  $(O_{c_i} - S_{c_i})$  objects may result in cache misses with probability  $(1 - \frac{B_{c_i}}{S_{c_i}})$  which

comes from the capacity limitation of page buffers, where  $B_{c_i}$  is the number of cache space allocated to the scan. Thus, the number of cache misses [Jea94] of the new design is as the following equation.

$$\begin{cases} Sc_i + (1 - \frac{Bc_i}{Sc_i}) \times (Oc_i - Sc_i), & \text{if } Bc_i < Sc_i \\ Sc_i, & \text{if } Bc_i \geq Sc_i \end{cases} \quad (3.3)$$

As for the miss ratio of page buffer at the second level, it is usually called page faults in conventional. The number of pages accessed by client for the scan of class  $C_i$  is at least  $Pc_i$  depend on whether the concurrency control is performed. It is assumed that the scan is running under the multiple transaction environment and the database sharing is full, that means all scans operate on the same database. We define a parameter  $\sigma$  that indicates the sharing factor of page buffer. Hence, the number of buffer misses (or page faults) are defined as  $(1 - \sigma)Pc_i$  approximately and the buffer miss ratio is  $1 - \sigma$ .

We use subscripts to indicate what level they are representing. It means that the system merely operates in in-page-class granularity and no lock de-escalation and callback overhead are evaluated because no concurrency control is conducted. We analyze the time to read an object based on the sequential scan operation instead of analyzing the write object operation.

$$AOAT_{withCache} = T_{hitCache} + MR_{L1} \times MP_{L1} \quad (3.4)$$

The miss penalty of the second level memory is expressed as follow:

$$MP_{L1} = T_{hitBuffer} + MR_{L2} \times MP_{L2} + (1 - MR_{L2}) \times T_{readGroup} + T_{transmit} \quad (3.5)$$

where  $MP_{L2} = T_{io} + T_{readGroup}$ . It means that the missed page is read from the disk and a group of objects are collected.  $T_{transmit}$  represents the time needed to transmit the entire group to the client.

We compare the above scheme with the pure object server architecture. The number of cache misses at the first level is  $Oc_i$  in sequential scan operation, each object access will result a cache miss and access the second level page buffer. As for the miss ratio  $MR_{L2}$  of the page buffer, the equation (3.3) can be also applied in the second level

page buffer, except for  $Bc_i$  is replaced by  $ServerBufSize$ . The above equations (3.4) (3.5) are modified into the following for pure object server.

$$AOAT_{withCache} = T_{hitCache} + MP_{L1} \quad (3.6)$$

$$MP_{L1} = T_{hitBuffer} + MR_{L2} \times MP_{L2} + (1 - MR_{L2}) \times T_{readObj} + T_{transmit} \quad (3.7)$$

Where  $MP_{L2} = T_{io} + T_{readObj}$ . The experiments begin with the analysis of new scheme that includes the miss ratio of object cache and the AOAT. Then we will compare the new scheme with the pure object server. Lastly we give a figure to show the improved ratio of the two schemes. The set of parameters in the experiment as shown in Table 3.1 are in terms of some publications [Agra85, Care94] and the characteristics of our system.

Parameter	Description	Value
<i>DatabaseSize</i>	Size of database in pages	5000 pages (20M)
<i>PageSize</i>	Size of a page	4096 bytes
<i>ObjectsPerPage</i>	Number of objects in a page	20 objects
<i>ClassPerDatabase</i>	Number of classes in a database	10 classes
<i>Oc<sub>i</sub></i>	Number of objects with class $C_i$	10000 objects
<i>PageLocality</i>	Number of objects scanned per page	4, 8, 12 objects per page
<i>Pc<sub>i</sub></i>	Number of pages which contain objects with class $C_i$	800, 1250, 2500 pages
<i>Bc<sub>i</sub></i>	Number of pages allocated to the scan at the client	5% to 25% of DB size
<i>ServerBufSize</i>	Server buffer size	50% of DB size, 2500 pages
<i>T<sub>hitCache</sub></i>	Time to hit the object cache at client	0.5 ms
<i>T<sub>hitBuffer</sub></i>	Time to hit the page buffer at server	0.5 ms
<i>T<sub>readObject</sub></i>	Time to read an object in a page	0.6 ms
<i>T<sub>readGroup</sub></i>	Time to read a group of objects with the same class within a page	5 ms
<i>NetworkBandwidth</i>	Network bandwidth	80 Mbits per second
<i>T<sub>transmit</sub></i>	Time to transmit the previous read group	0.4 ms
<i>T<sub>io</sub></i>	Average disk page access time	20 ms for average
<i>T<sub>lock</sub></i>	Time to acquire a lock	0.5 ms
$\sigma$	Sharing factor of page buffers at server	10% to 90%

Table 3.1. The parameters of cost model

### 3.2. Experiment results and analysis

The results of the experiments are shown in the following figures and we will make some discussions about these observations.

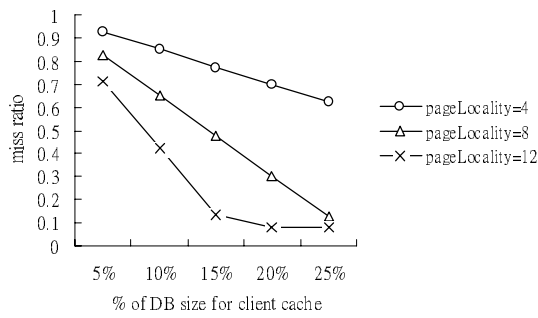


Fig. 3.1 Miss ratio under different % of DB size.

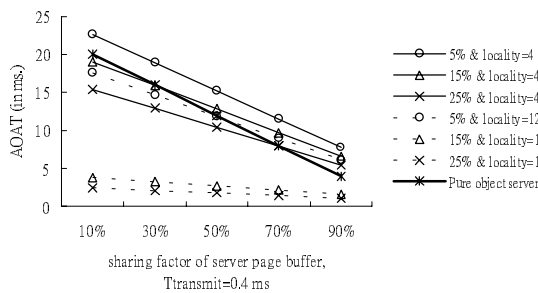


Fig. 3.2(a) AOAT under different  $\sigma$ .

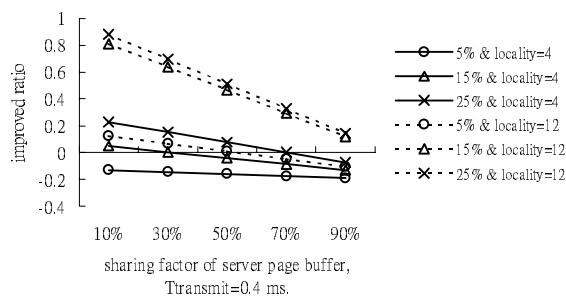


Fig. 3.2(b) Improved ratio under different  $\sigma$ .

$$\text{Improved ratio} = (\text{AOAT}_{\text{pure}} - \text{AOAT}_{\text{new scheme}}) / \text{AOAT}_{\text{pure}}$$

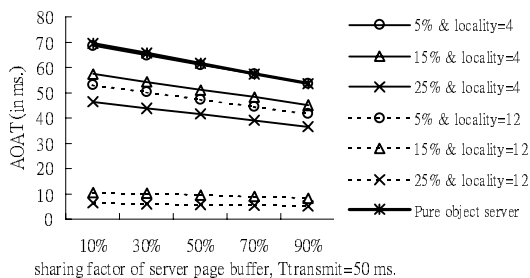


Fig. 3.3(a) AOAT under different  $\sigma$ .

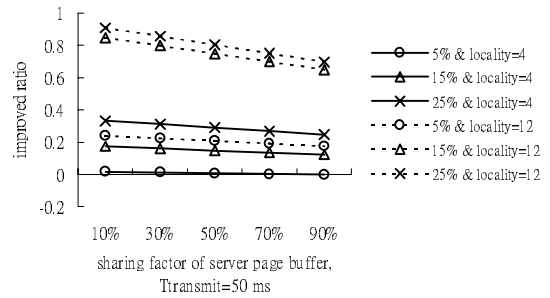


Fig. 3.3(b) Improved ratio under different  $\sigma$ .

$$\text{Improved ratio} = (\text{AOAT}_{\text{pure}} - \text{AOAT}_{\text{new scheme}}) / \text{AOAT}_{\text{pure}}$$

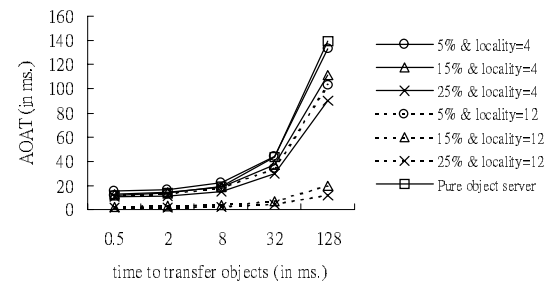


Fig. 3.4 AOAT under different transmission time.

- (1) The higher the page locality is clustered, the lower the miss ratio is found. When the allocated cache space at client is larger than the actual touched pages, the miss ratio will reach a good constant value as shown at the bottom in Figure 3.1.
- (2) From Figure 3.1, a larger client cache setting will make a significant difference about miss ratio. This indicates that more cache space at client will effectively reduce the miss ratio. Moreover, the higher of page locality makes a faster decreasing ratio about the cache miss.
- (3) From Figure 3.2(a), some observations are made. First, the higher the sharing factor of page buffers at server are set, the closer the access time results. Second, the lower page locality one is more sensitive with the sharing factor of page buffers than the higher one. Finally, the one with the higher page locality has the lower average object access time.
- (4) As for the comparison of the new scheme and the

pure object server, the object access time of pure object server is faster in all cases when sharing factor is greater than 70% and the page locality is less than 4 as shown in Figure 3.2(a). Second, when the page locality equals 12 or higher and the cache space equals 15% of database size or larger, the new scheme will always wins over the pure object server and get to a fairly good performance. Moreover, when the sharing factor of server page buffer is low, almost all the cases of new scheme will win as shown in Figure 3.2(a) and Figure 3.2(b). Finally, when the sharing factor of server page buffer is greater than 70% and the page locality is 4, the pure object server wins in all cases. This indicates that the performance of pure object server will be better than the new scheme under poor clustering pages and the high multiple programming level.

- (5) From Figure 3.3(a), Figure 3.3(b) and Figure 3.4, we observe the effects of communication speed. When the network speed is slow, the new scheme under all circumstances dominates the pure object server. At the same time, we also find out that the access time is always good regardless of the network speed while the page locality is high.

#### 4. Conclusion and future research

CACTUS, an object oriented database system, based on the peer-to-peer architecture is newly designed and partly implemented at the current stage. Some components are still under developing according to our plan. It has the distributed transaction processing ability under multi-thread environment and is capable of maintaining the database consistency in case of system failures occur.

In this article, we present a new scheme to improve the performance of object server and increase its degree of

concurrency. The presence of in-page-class granularity and its associated lock result better performance than object server and produce lower probability of lock conflict than page server. The current architecture implemented of CACTUS is based on a pure object server with a simple object cache at client. In ours experiments, some observations and conclusions about the new scheme are made and they are valuable for future references. Finally, we conclude that the new scheme is worth fulfilling to accelerate the execution.

#### References

- [Agra85] R. Agrawal, D. J. Dewitt, "Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 10, No. 4, pp. 529-564, Dec. 1985.
- [Care94] M. J. Carey, M. J. Franklin, M. Zaharioudakis, "Fine-Grained Sharing in a Page Server OODBMS," *SIGMOD 94*, Minneapolis, Minnesota, USA, pp. 359-370, 1994.
- [Jea94] K. F. J. Jea, S. Y. Chen, Y. T. Lee, "Buffer Management for Object-Oriented Database Management Systems," *Proceedings of International Computer Symposium 1994*, Hsinchu, Taiwan, pp. 1164-1171, December, 1994.
- [Moha92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," *ACM Transactions on Database Systems*, Vol. 17, No. 1, pp. 94-16