Performance Evaluation of Combining Data Migration and Method Migration in Object Database Environments

Yusuke Yoshida **Masayoshi Aritsugi** **Yoshinari Kanamori**

Department of Computer Science, Faculty of Engineering, Gunma University
1-5-1 Tenjin-cho, Kiryu 376-8515, Japan
Email: {yosida@dbms.,aritsugi@,kanamori@}cs.gunma-u.ac.jp

Abstract

In this paper, we discuss efficient combinations of data migration processing and method migration processing in object database environments. When we have multiple servers and clients, it is practical to decide at a client which of the two ways should be adopted for performing each processing with a server for performance. This paper presents an analytical model of the two ways in object database environments. In addition, we report experimental results showing that the model enables us to get the efficient combination of them in most cases.

Keywords: Distributed processing; Object databases; Performance evaluation; Distributed methods; Data migration; Method migration

1 Introduction

Advances in computer technology have made it possible to have environments comprising many powerful workstations and personal computers that are connected by a network. It is therefore important to study ways for obtaining better performance of processing in distributed environments.

There are mainly two ways to perform a method to datasets in distributed environments: data migration and method migration. However, only the way of data migration processing has so far been adapted in object databases [Carey et al., 1994, Lamb et al., 1991, ONTOS Inc., 1994]. Several studies [Franklin et al., 1996, Goel et al., 1996] suggested that method migration should be exploited with data migration for improving performance of distributed database processing. Since there are a lot of combinations of data migration and method migration in environments having multiple servers and multiple clients, it is an challenging issue how to perform each remote processing between a client and a server. We may often see such situations in the real world. For example, an employee database tends to consist of several datasets, each of which is stored and managed by different department in a company, and the datasets are shared and accessed by departments randomly. In this paper, we discuss efficient combination of data migration and method migration processes. We propose an analytical model for processing in distributed environments, and compare it with experimental results showing that the model works well in most cases.

For method migration in object database environments, we do not assume that a method implementation, or a method code to be processed, is on every server site before processing; it is moved from a client to server whenever it is required. If it is essential that method implementation should be on sites where the method might be performed in advance of processing, the cost for doing that would be large. Rodríguez-Martínez and Roussopoulos [Rodríguez-Martínez and Roussopoulos, 2000] wrote that it is unrealistic to assume that every site has necessary operators that are already implemented before processing in the context of large-scale distributed environments. Gray and Shenoy [Gray and Shenoy, 2000] insisted that it is very important to reduce database administrative costs. Due to these observations, we have developed an environment where method codes are stored and managed by an object database and method processing is performed by retrieving the method code to be processed from the database and transmitting it to the site at which it is performed [Aritsugi et al., 2000], if necessary. In the environment we do not have to assume that the method implementation to be processed should exist at the site where it will be performed in advance.

In several studies on query processing in object databases, a query, or a method, is decomposed into primitive operators, and how to perform the operators is discussed [Franklin et al., 1996, Özsu and Blakeley, 1995]. On the other hand, we study optimization without decomposing a method in this paper. Since in object database environments the internal structure of objects is usually invisible [Özsu and Blakeley, 1995], we think it is necessary to study on optimization without the decomposition. The results of this kind of studies can be applied for query processing using stored procedures. Note that we do not intend to propose a new optimization technique that overcomes conventional optimization techniques. Rather, the results of this paper can work with conventional optimization techniques.

The remainder of this paper is organized as follows. Section 2 compares related work with ours for clarifying our contributions of this study. Section 3 briefly explains environments on which the following discussions are based. Section 4 studies combinations of method migration processing and data migration processing in object database environments. Section 5 presents estimation results and experimental results, and conclusions are given in Section 6.

2 Related Work

It is very important to study performance improvement in distributed database processing. Most of relational database systems and object database systems have supported method migration and data migration, respectively. However, as Franklin *et al.* [Franklin et al., 1996] wrote, we lack studies on performance improvement by means of combining the

two migration techniques.

Franklin *et al.* [Franklin et al., 1996] discussed query processing by means of data shipping, query shipping, and hybrid shipping techniques. They decomposed accesses to object databases into unary operations (e.g., selection) and binary operations (e.g., join), and applied conventional approaches developed for relational databases using the three shipping techniques to query processing in object databases. On the other hand, in this paper we study on query processing without the decomposition.

Rodríguez-Martínez and Roussopoulos [Rodríguez-Martínez and Roussopoulos, 2000] studied on tradeoffs between data shipping and code shipping in client-server database system environments. They proposed a method for deciding which of the two processing ways should be used. The method is based on the characteristic of an operator whether the size of data grows larger due to the application of the operator. The main differences between their study and ours are (1) their study was based on an environment in which there were only single server and single client, and (2) they did not take into account of the loads on server caused by other clients in a network.

Goel *et al.* [Goel et al., 1996] studied on the performance of method migration and data migration in client-server database systems. They did experiments with varying the size of objects using single server and single client in environments of LAN, MAN, and WAN. The results showed that in cases where the large size of objects are used and a large amount of objects have to be transmitted in the network, or the network bandwidth is narrow, method migration would be useful for performance improvement. The main different point between their study and ours is that we take multiple servers and multiple clients into account in this study.

3 A Distributed Database Environment

In this section we describe briefly the object database environment we have developed in which method migration can be supported; a more detailed description including implementation details can be found in [Aritsugi et al., 2000]. This environment was used for experiments reported in this paper.

We have developed the environment in Java and ObjectStore [Lamb et al., 1991]. Nevertheless to say, object database management systems can handle complex data more easily than relational database management systems. By making use of this ability in distributed environments, we can store methods into an object database. This allows us to implement a prototype system of *distributed methods*. We can realize method migration by using distributed methods. For performing a method, the code to be processed should be loaded dynamically to a process runnable at the site where the method is performed. We have implemented this dynamic loading facility with the function of ClassLoader in Java [Aritsugi et al., 2000].

Figure 1 represents the process structure for distributed methods. For simplicity, only three sites appear in the figure. Similar to SHORE [Carey et al., 1994], the structure consists of peer-to-peer servers and clients. A server must be running on each workstation on which users want to perform distributed methods. Every workstation must manage a database for distributed methods, which is referred to as MethodDB in the figure. DataDB in the figure stands for conventional databases; it stores and manages data members of objects.

A server process for distributed methods and a client process are both performed actually by threads in the implementation. If a data migration request and a method migration request are performed by respective threads in a process, these threads can be performed concurrently. On the other hand, threads for multiple method migration requests or multiple data migration requests in a process are performed sequentially.

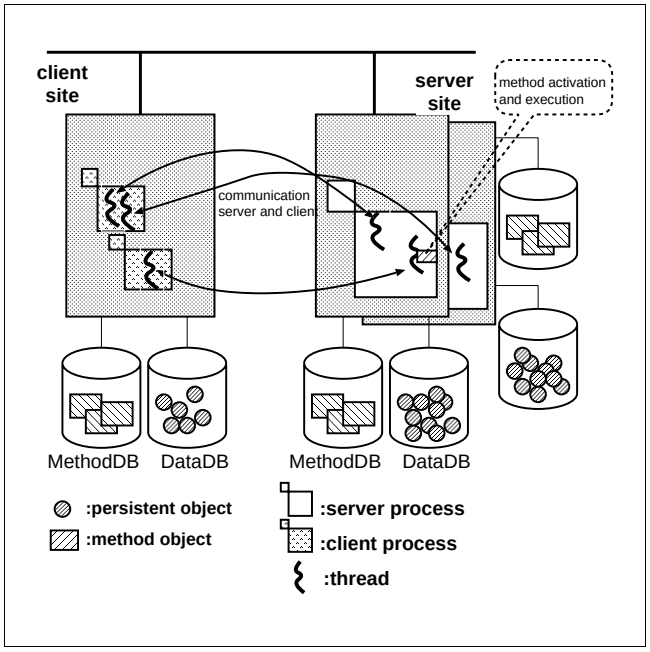

Figure 1: Process structure for distributed methods

4 Combination of Method Migration and Data Migration

Figure 2 shows an example of distributed database environments we consider in this paper. In such an environment in which there are multiple servers and multiple clients, a client has to decide which of method migration and data migration should be used for processing on each server when the client handles data managed by multiple servers. If a client processes data managed by n servers, then the number of possible combination patterns of method migration and data migration is 2^n. In this paper, we try to obtain the efficient combination by which the response time of the client is minimum, out of the 2^n. For simplicity, we assume that methods do not include update processing throughout this paper.

4.1 An Analytical Model

If a database consists of several datasets, each of which is managed by a site in a network, a process on the database is performed by multiple processes to the sites. For example, if an employee database is managed by each department, then a process on all employee data is performed by multiple processes to all the departments.

Let us assume that server site $S_i (1 \leq i \leq n)$ manages data of size D_{S_i} pages, and the size of the method to be processed is M pages. We express in this paper the network bandwidth between the server site and client site C as NW pages per second, the speed of disk I/O at S_i as DW_{S_i} pages per second, the speed of disk I/O at C as DW_C pages per second, the size of data that C can process in a second as PT_C pages, and the size of data that S_i can process in a second as PT_{S_i} pages. As shown in Figure 2, every site is connected by a switch, and we assume there is no network saturation.

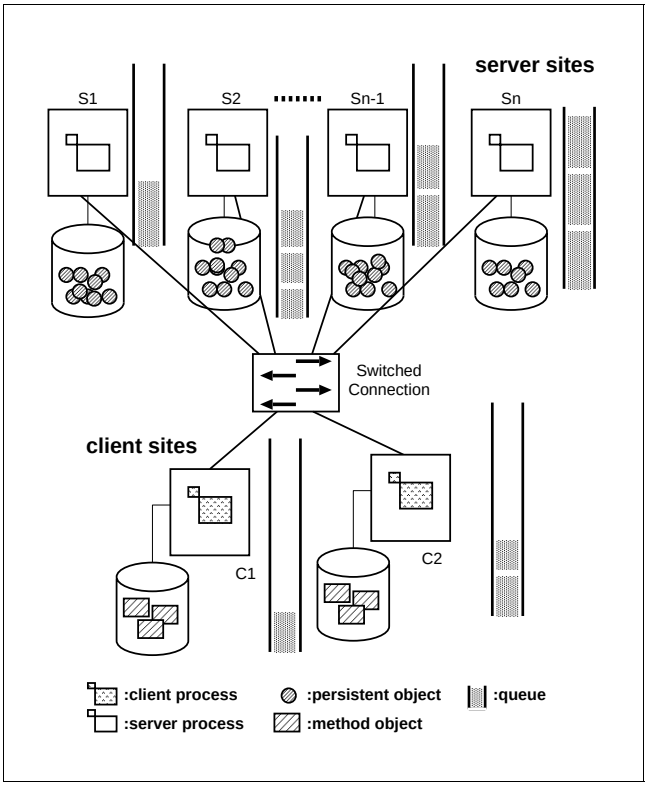

Figure 2: An example of distributed database environments

The initialization cost of methods, T_M, including the cost for obtaining the method implementation from MethodDB, transmitting it to the site where it is performed, and activating the method code, is expressed as

$$T_M = M \times \frac{1}{DW} + M \times \frac{1}{NW} + AC \qquad (1)$$

where DW is DW_C if the method implementation to be processed is stored at the client, or DW_{S_i} if that is stored at the server site S_i, and AC is the cost for activating the method code. The second term is 0 when there is no need to transmit the method implementation, or when the method implementation has already existed at the site where it is performed.

Let us first consider a case in which we have a server site S_1 and a client site C. The response time, T_C, for processing data on S_1 by data migration is expressed as

$$T_C = T_M + D_{S_1} \times (\frac{1}{DW_{S_1}} + \frac{1}{NW} + \frac{1}{PT_C}). \qquad (2)$$

On the other hand, the response time, T_{S_1}, for processing by method migration is expressed as

$$T_{S_1} = T_M + D_{S_1} \times (\frac{1}{DW_{S_1}} + \frac{1}{PT_{S_1}} + \frac{f}{NW}) \qquad (3)$$

where $f (0 \leq f \leq 1)$ is used for expressing the size of returned data as $f \times D_{S_1}$.

Then, let us extend these considerations to environments in which we have multiple servers and multiple clients. Due to multiple occurrences of method migration requests from clients to a server, each request might wait to be performed at the server site until all other requests that had reached the server previously are processed. When a client site processes on data managed by multiple servers by data migration, these data migration processes are performed sequentially.

Let us model the load of a server site caused by multiple method migration requests. We estimate disk I/O, method processing, and transmission through a network as a service, and calculate mean service rate by means of mean response time of a request from a client. We therefore model the environment with the M/M/1 system in this paper.[1] We assume that requests from clients arrive at a server randomly, and the service time distribution has the exponential distribution. Let λ be mean arrival rate and μ be mean service rate. The load rate at a server site is expressed as

$$\rho = \frac{\lambda}{\mu}. \qquad (4)$$

In the M/M/1 system, mean queue length L is expressed as $\sum_{n=1}^{\infty}(n-1)\rho = \frac{\rho^2}{1-\rho}$.[Sauer and Chandy, 1981] Let W be mean waiting time. We know $L = \lambda W$ as Little's Rule. Then, W is expressed as

$$W = \frac{1}{\lambda} \times \frac{\rho^2}{1-\rho}. \qquad (5)$$

Let us consider the processing at a client, i.e., receiving data from servers and processing them in data migration processing, and receiving results from the other servers in method migration processing. Figure 3 shows the parallel processing model[2] at the client. In the figure, $T_{parallel}[i]$ stands for the cost at server site S_i of processing a request from the client independently from the other sites, and $T_{serial}[i]$ stands for the cost at the client site of processing sequentially. We sort the set of servers $\{S_1, \cdots, S_n\}$ in order of the value $T_{parallel}[i]$ in the figure, so that we can assume $T_{parallel}[1] \leq \cdots \leq T_{parallel}[n]$.

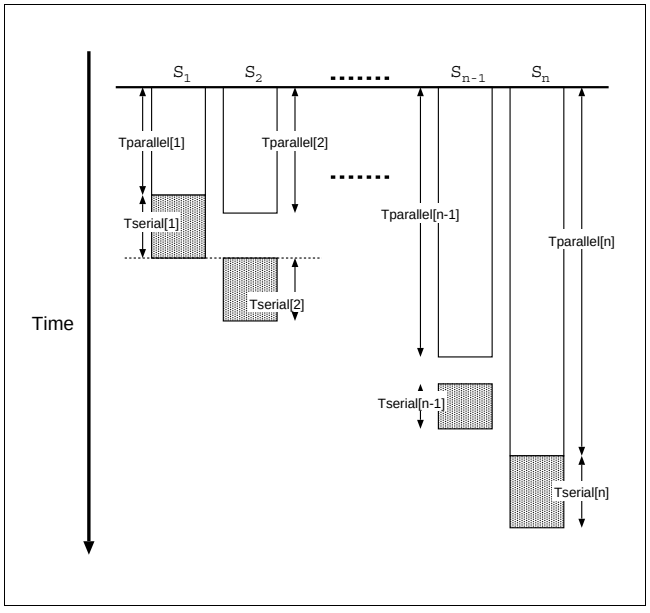

Figure 3: Parallel processing model

In data migration processing, $T_{parallel}[i]$ includes the cost expressed by eq. (1) and of disk I/O at server site i, and $T_{serial}[i]$ includes transmitting data and

[1] In this paper, we consider a method of objects as a scheduling unit. It is generally difficult to split a method into read and write operations as in the transactional processing field.

[2] A similar model can be found in [Aritsugi et al., 2001].

performing a method to the data at the client. In method migration processing, $T_{parallel}[i]$ includes the cost expressed by eq. (1), that of mean waiting time expressed by eq. (5), that of disk I/O at each server site, and that of performing the method at the site. $T_{serial}[i]$ in this processing is the cost for transmitting results from server site S_i to the client.

Let $N = \{1, 2, \cdots, n\}$, $DM \subseteq N$, $MM \subseteq N$ $(DM \cap MM = \phi \wedge DM \cup MM = N)$, $\{S_k | k \in N\}$ be the set of server sites, $\{S_i | i \in DM\}$ be the set of server sites on each of which data are processed by data migration, and $\{S_i | i \in MM\}$ be the set of server sites on each of which data are processed by data migration. Then, based on eqs. (2) and (3), $T_{parallel}[i]$ and $T_{serial}[i]$ are expressed as follows.

$$T_{parallel}[i] = \begin{cases} T_M + D_{S_i} \times \dfrac{1}{DW_{S_i}} & (i \in DM) \\ T_M + W & \\ \quad + D_{S_i} \times (\dfrac{1}{DW_{S_i}} + \dfrac{1}{PT_{S_i}}) & (i \in MM) \end{cases} \tag{6}$$

$$T_{serial}[i] = \begin{cases} D_{S_i} \times (\dfrac{1}{NW} + \dfrac{1}{PT_C}) & (i \in DM) \\ D_{S_i} \times \dfrac{f}{NW} & (i \in MM) \end{cases} \tag{7}$$

We can estimate the response time of the client, T, with eqs. (6) and (7) by the algorithm shown in Figure 4, which is derived from the parallel processing model shown in Figure 3. In Figure 4, t_p and t_s correspond to $T_{parallel}[i]$ and $T_{serial}[i]$, respectively.

```
function ResponseTime(DM, MM)
begin
    var S : an empty list;
    foreach i ∈ DM ∪ MM do
    begin
        if i ∈ DM then begin
            t_p ← T_M + D_{S_i} × 1/DW_{S_i};
            t_s ← D_{S_i} × (1/NW + 1/PT_C)
        end
        else begin
            t_p ← T_M
                  +W + D_{S_i} × (1/DW_{S_i} + 1/PT_{S_i});
            t_s ← D_{S_i} × f × 1/NW;
        append the pair (t_p, t_s) into S
        end
    end
    sort S by t_p;
    T ← 0;
    foreach (t_p, t_s) in S do
        if T > t_p then
            T ← T + t_s
        else
            T ← t_p + t_s;
    end
    return T
end ;
```

Figure 4: The algorithm for the response time T

4.2 A Strategy for the Efficient Combination

In order to obtain the most efficient combination, we consider the tradeoff between parallel processing by method migration processing and reducing the load of servers by data migration processing. The computing powers of server sites are generally larger than those of client sites. When a client wants to process data on multiple server sites, each method migration processing can be performed in parallel, while each data migration processing must be performed sequentially at the client site. Thus, the processing in which as many data as possible are processed by method migration would improve the performance. Note, however, that we have to take care that method migration processing may make the performance worse when the load of the server is high. According to these observations, we propose an algorithm for obtaining the most efficient combination of method and data migration processing, which is described in Figure 5.

```
function EfficientCombination(N)
begin
    {Let us assume T_parallel[i] ≤ T_parallel[j]
                        (i, j ∈ N; i < j)}
    DM ← N; MM ← φ;
    T' ← ResponseTime(DM, MM);
    T ← ResponseTime(DM−{1}, MM∪ {1});
    k ← 2;
    while T ≤ T' and k ≤ n do
    begin
        DM ← DM − {k};
        MM ← MM ∪ {k};
        T'←T;
        T ← ResponseTime(DM, MM);
        k ← k + 1
    end
    if T ≥ T' then
        DM ← DM ∪ {k};
        MM ← MM − {k};
    end
    return (DM, MM)
end ;
```

Figure 5: The algorithm for efficient combination

The algorithm is designed based on the following ideas. We first assume that processes to data on S_1, ..., S_n are all performed by data migration processing. The response time T under this assumption is calculated with the algorithm shown in Figure 4 by setting $DM = N$ and $MM = \phi$. Then, we try to improve the performance by changing some data migration processing to method migration processing, in which action we select the data migration processing with the server site having the least value of $T_{parallel}[i]$.

5 Performance Evaluation

5.1 An Experimental Environment

For the experiments we used four workstations, the configuration of which is shown in Table 1. The workstations were connected by a 100 Mbps Ethernet. Sites S_1, S_2, and S_3 were used as server sites, and site C as a client site. During the experiments, we excluded other processes from the environment.

We created sets of `Person` objects on the server workstations. An object with `name`, `age`, `salary`

Table 1: Testbed configuration

site	S_1, S_2, S_3	C
machine type	Sun Ultra 30	Sun Ultra 1
cpu	UltraSPARC-II	UltraSPARC-I
clock	296MHz	167MHz
memory size	128MB	128MB
disk	SUN4.2G	SUN2.1G
page size	8192(bytes)	
JavaVM	JDK 1.1.6(native thread)	
DBMS	ObjectStore R5.1/ Java Interface R3.0	

had `x` and `image` as its members. The member `x` was used as a dummy attribute, and `image` had a bitmap image data of size 2 Kbytes. The number of the attributes came from the `Part` object in the OO1 Benchmark [Cattell and Skeen, 1992]: `to` and `from` attributes appearing in OO1 were ignored because they were list attributes. The `age` value was an integer and randomly distributed in the range [0..99] at each server site. We generated 5000 objects at each server site. The size of them at each server site was about 10 Mbytes. The `salary` value was generated using the `age` value. We implemented set objects using a class library provided by ObjectSpace Inc.[ObjectSpace Inc., 1997] The set object was a listed set in which `Person` objects were linked only in creation order with no indexing facility. The method used in the experiments took an `age` value as its argument and calculated the average `salary` of `Person` objects under the `age` value. We assumed that the method code had been developed at the client site, and the code had not been stored at any server site but at the client site because of the description in [Rodríguez-Martínez and Roussopoulos, 2000].

We set μ in eq.(4) 0.092 according to measured values. Server sites' waiting time was realized using function `java.lang.Thread.sleep()` in Java.

In the experiments we assumed that various applications were performed at multiple client sites and data were thus accessed in various ways. In addition, the program scanned whole databases every run in the experiments so that we could not exploit the cache at the client site effectively, since ObjectStore manages cache according to the LRU algorithm. We therefore do not take the effect of caches into account in this study.

5.2 Results

In the experiments we varied ρ in eq.(4) at each of the three servers as shown in Table 2. We modeled the load of a server as three types, i.e., low, high, and middle, by setting ρ 0.2, 0.5, and 0.8, respectively. Incidentally, we confirmed that if we set the value of ρ over 0.8 or under 0.2 the results were not so different from those when the value was 0.8 or 0.2.

Table 2: ρ

site	S_1	S_2	S_3
L(Low loads)	0.2	0.2	0.2
M(Mixed loads)	0.2	0.5	0.8
H(High loads)	0.8	0.8	0.8

Table 3 shows the parameter settings we used in the estimation. These values were measured in the environment where we run the experiments by assuming that the amount of data accessed by dummy clients

Table 3: Parameter settings

site	S_1, S_2, S_3	C
$D_{site}(pages)$	1283(≃10MB)	
$NW(pages/sec)$	273.6(≃2189KB)	273.6(≃2189KB)
$DW_{site}(pages/sec)$	131.3(≃1050KB)	131.3(≃1050KB)
$PT_{site}(pages/sec)$	928.0(≃7.3MB)	520.0(≃4.1MB)
AC	0	0

was randomly distributed in the range $[0 \times D_{S_i}..1 \times D_{S_i}]$. The cost for activating the method code is set to 0, because the cost could be ignored compared with other values, due to the small size of the method implementation code we used in the experiments. For simplicity we fixed these values shown in Table 3 for the estimation.

In the following we express results with three characters of `d` or `m`: `d` corresponds to data migration processing and `m` corresponds to method migration processing. For example, `mdm` expresses that the client performed the experimental program with servers S_1, S_2, and S_3 by method migration processing, data migration processing, and method migration processing, respectively.

Table 4: The most efficient combinations in the estimation

f	0	0.1	0.2	0.3	0.4	
L	`mmm`	`mmm`	`mmm`	`mmm`	`mmm`	
M	`mmd`	`mmd`	`mmd`	`mmd`	`mmd`	
H	`ddd`	`ddd`	`ddd`	`ddd`	`ddd`	
f	0.5	0.6	0.7	0.8	0.9	1
L	`mmm`	`mmm`	`mmm`	`mmm`	`mmm`	`mmm`
M	`mmd`	`mmd`	`mmd`	`mmd`	`mmd`	`mmd`
H	`ddd`	`ddd`	`ddd`	`ddd`	`ddd`	`ddd`

Table 4 shows the estimation results using the values shown in Tables 2 and 3. When all the three server sites were about idle, the model estimated that `mmm` had the best performance. On the other hand, when all server sites were in high load, it estimated that the combination `ddd` gave the best performance. In the case of M in Table 2, it suggested that data migration processing should be selected when processing on data stored at server S_3 because its load was high.

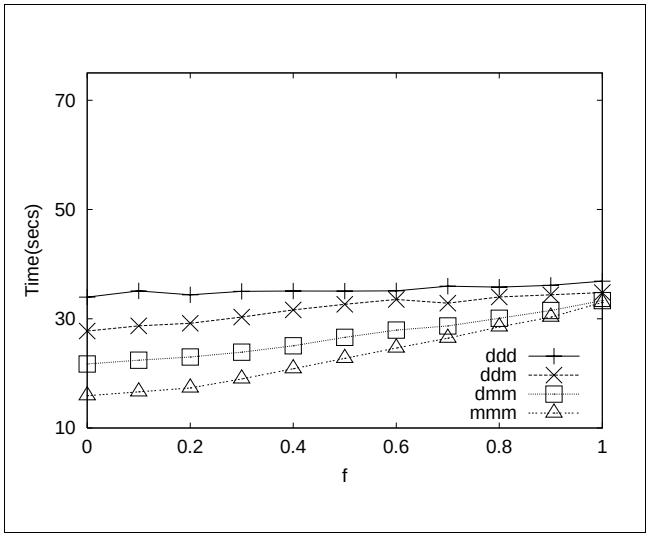

Figure 6: Experimental results: L

Figure 6 shows the results of the case where all the three server sites were about idle. In this case, the

combinations of ddm, dmd, and mdd and those of dmm, mdm, and mmd were identical; the figure thus shows the results of ddd, ddm, dmm, and mmm. As shown in the figure, mmm had the least response time in the experiments. We can see from the figure that the performance of ddd was about constant regardless of the value of f, while it was the worst in this case. The performance of ddd was larger than that of mmm when $f = 1$ due to the difference between the computing powers of server sites and the client site.

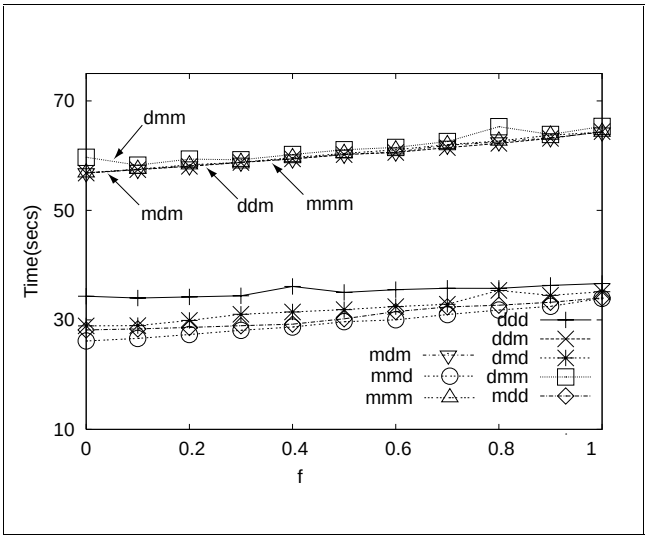

Figure 7: Experimental results: M

Figure 7 shows the results of M in Table 2. In the combinations of ddm, dmm, mdm and mmm, the processing was performed at server site S_3, which had the highest load. These combinations therefore gave the worst performance in this case. The combination mmd realized the best performance in this case.

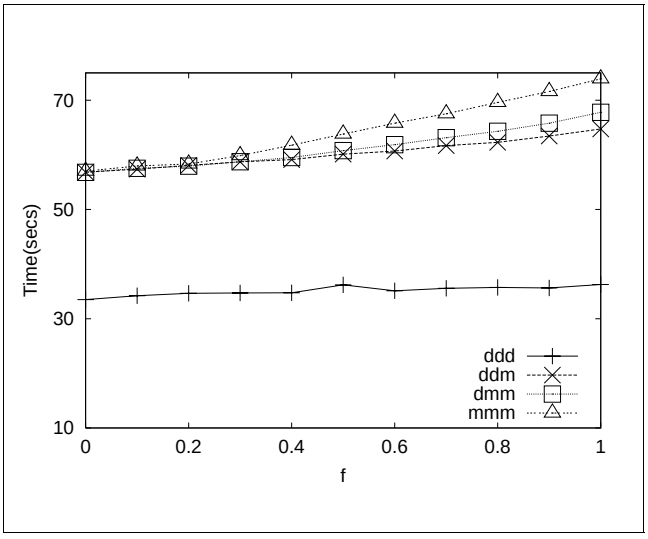

Figure 8: Experimental results: H

Figure 8 shows the results of the case where all server sites were in high load. Similar to the case of L, the figure shows the results of ddd, ddm, dmm, and mmm. The combinations of ddm, dmm, and mmm gave worse performances due to the bottleneck of method migration processing at the servers. The combination ddd gave the best performance in this case.

By comparing the experimental results shown in Figures 6, 7, and 8, and the estimation results shown in Table 4, the analytical model could estimate the most efficient combinations of method migration processing and data migration processing in all cases in the experiments.

Next we show another experiments and results. In the following experiments, we set $f = 1$ and varied ρ at each of the three server sites. Let ρ_1, ρ_2, and ρ_3 be the load rate at server site S_1, S_2, and S_3, respectively. We run experiments with the fixed value of f while varying ρ_1, ρ_2, and ρ_3 as shown in Table 5.

Table 5: Load rates for R2, R5, and R8

site	S_1	S_2	S_3
R2	0.2	[0.2..0.8]	[0.2..0.8]
R5	0.5	[0.2..0.8]	[0.2..0.8]
R8	0.8	[0.2..0.8]	[0.2..0.8]

Figures 9, 10, and 11 show the results and error rates in cases of R2, R5, and R8 in Table 5, respectively, where error rate is defined as follows.

$$\frac{(resp.\ time\ of\ est.) - (resp.\ time\ of\ exper.)}{resp.\ time\ of\ exper.} \quad (8)$$

For example, when $(\rho_1, \rho_2, \rho_3) = (0.2, 0.5, 0.4)$ the estimated combination was mmm, and the most efficient combination obtained from the experiments was mdm, as shown in Figures 9(a) and 9(b), respectively. The response times of mmm and mdm were obtained from the experiments as 27.27 and 26.53, respectively. Thus, $(resp.\ time\ of\ est.) - (resp.\ time\ of\ exper.) = 0.74$ and the error rate in this case was $0.74/26.53 \simeq 0.03$, as shown in Figure 9(c). In each (c) figure, "-" means that the combination obtained from the experiments and the estimation was identical in the case.

We can see from the figures that the analytical model can work well in many cases; in fact, the estimation results were the same as the experimental results in 121 out of 147 combinations of (ρ_1, ρ_2, ρ_3). Note that the median of frequency distribution of error rates was 0.03; that is, the model could predict the most efficient combination of method migration and data migration in 95.2% cases with less than or equal to 0.03 error rate.

Figure 12 shows the correlation between average of load rates and error rate. We can see two correlations in the figure. When average of load rates is about in the range [0.3..0.46], error rate is under 0.04; thus, the estimation is useful practically. On the other hand, when average of load rates is about in the range [0.56..0.73], error rate is relatively large ([0.03..0.06]); this is because the analytical model predicted that method migration processing would improve the performance although data migration processing should be selected in the experiments. We think that the current analytical model tends to estimate the cost for data migration processing greatly. As shown in the paper, we consider that at the client site receiving data from a network and performing a method are done sequentially. However, these two processes might be slightly overlapped in practical situations. In a future work we will attempt to extend our modeling method to overcome this problem.

6 Conclusions

We studied combinations of method migration and data migration in object database environments for performance. We presented the analytical model of these two processing methods in object database environments. In this paper, we also presented experi-

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	mmm	mmm	mmm	mmm	mmm	mmd	mmd
0.3	mmm	mmm	mmm	mmm	mmm	mmd	mmd
0.4	mmm	mmm	mmm	mmm	mmm	mmd	mmd
0.5	mmm	mmm	mmm	mmm	mmm	mmd	mmd
0.6	mmm	mmm	mmm	mmm	mmd	mmd	mmd
0.7	mdm	mdm	mdm	mdm	mdd	mdd	mdd
0.8	mdm	mdm	mdm	mdm	mdd	mdd	mdd

(a) Estimation results

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	mmm	mmm	mmm	mmd	mmd	mmd	mmd
0.3	mmm	mmm	mmm	mmd	mmd	mmd	mmd
0.4	mmm	mmm	mmm	dmm	mmd	mmd	mmd
0.5	mmm	mmm	mdm	dmm	mmd	mmd	mmd
0.6	mdm	mdm	mdm	mdm	mmd	mmd	mmd
0.7	mdm	mdm	mdm	mdm	mdd	mdd	mdd
0.8	mdm	mdm	mdm	mdm	mdd	mdd	mdd

(b) Experimental results

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	–	–	–	0.03	0.02	–	–
0.3	–	–	–	0.01	0.01	–	–
0.4	–	–	–	0.02	0.00	–	–
0.5	–	–	0.03	0.00	0.01	–	–
0.6	0.02	0.03	0.02	0.02	–	–	–
0.7	–	–	–	–	–	–	–
0.8	–	–	–	–	–	–	–

(c) Error rates

Figure 9: R2

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	mmm	mmm	mmm	mmm	mmd	mmd	mmd
0.3	mmm	mmm	mmm	mmm	mmd	mmd	mmd
0.4	mmm	mmm	mmm	mmm	mmd	mmd	mmd
0.5	mdm	mdm	mdm	mdm	mmd	mmd	mmd
0.6	mdm	mdm	mdm	mdm	mmd	mmd	mmd
0.7	mdm	mdm	mdm	mdm	mdd	mdd	mdd
0.8	mdm	mdm	mdm	mdm	mdd	mdd	mdd

(a) Estimation results

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	mmm	mmm	dmm	mmd	mmd	mmd	mmd
0.3	mmm	mmm	dmm	dmm	mmd	mmd	mmd
0.4	mmm	mmm	dmm	dmm	mmd	mmd	mmd
0.5	mdm	mdm	mdm	mdm	mmd	mmd	mmd
0.6	mdm	mdm	mdm	mdm	mmd	mmd	mmd
0.7	mdm	mdm	mdm	mdm	mdd	mdd	mdd
0.8	mdm	mdm	mdm	mdm	mdd	mdd	mdd

(b) Experimental results

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	–	–	0.04	0.01	–	–	–
0.3	–	–	0.01	0.01	–	–	–
0.4	–	–	0.01	0.01	–	–	–
0.5	–	–	–	–	–	–	–
0.6	–	–	–	–	–	–	–
0.7	–	–	–	–	–	–	–
0.8	–	–	–	–	–	–	–

(c) Error rates

Figure 10: R5

mental results obtained from a real network of workstations in which not only data migration processing but also method migration processing were implemented using an object database. The experimental results showed that the analytical model can work well in many practical situations.

In a future work, we will attempt to integrate this work with conventional optimization techniques in which a query, or a method, is decomposed into primitive operators, namely, unary and binary operators. In the present study, we assumed that methods did not include update processing. We will attempt to extend this work to applications of methods including update processing.

References

[Aritsugi et al., 2001] Aritsugi, M., Fukatsu, H., and Kanamori, Y. (2001). Several partitioning strategies for parallel image convolution in a network of heterogeneous workstations. *Parallel Computing*, 27(3):269–293.

[Aritsugi et al., 2000] Aritsugi, M., Yoshida, Y., Nakamura, T., and Kanamori, Y. (2000). Method migration in object database environments. In *4th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2000/ISAS 2000)*, volume VIII, pages 125–130, Orlando, USA.

[Carey et al., 1994] Carey, M.J., DeWitt, D.J., Franklin, M.J., Hall, N.E., McAuliffe, M.L., Naughton, J.F., Schuh, D.T., Solomon, M.H., Tan, C.K., Tsatalos, O.G., White, S.J., and Zwilling, M.J. (1994). Shoring up persistent applications. In *ACM SIGMOD International Conference on Management of Data*, pages 383–394, ACM Press.

[Cattell and Skeen, 1992] Cattell, R.G.G. and Skeen, J. (1992). Object operations benchmark. *ACM Trans. Database Syst.*, 17(1):1–31.

[Franklin et al., 1996] Franklin, M.J., Jónsson, B.T., and Kossmann, D. (1996). Performance tradeoffs for client-server query processing. In *ACM SIGMOD International Conference on Management of Data*, pages 149–160, Montréal, Canada. ACM Press.

[Goel et al., 1996] Goel, S., Bhargava, B., and Jiang, Y.H. (1996). Supporting method migration in a distributed object database system: a performance study. In *IEEE 29th Annual Hawaii International Conference on System Sciences*, pages 31–40.

[Gray and Shenoy, 2000] Gray, J. and Shenoy, P. (2000). Rules of thumb in data engineering. In *16th International Conference on Data Engineering*, pages 3–10.

[Lamb et al., 1991] Lamb, G., Landis, G., Orenstein, J., and Weinreb, D. (1991). The ObjectStore database system. *Commun. ACM*, 34(10):51–63.

[Rodríguez-Martínez and Roussopoulos, 2000] Rodríguez-Martínez, M. and Roussopoulos, N. (2000). MOCHA: A self-extensible database middleware system for distributed data sources. In *ACM SIGMOD International Conference on Management of Data*, pages 213–224, Dallas, USA. ACM Press.

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	dmm	dmm	dmm	dmm	dmm	dmd	dmd
0.3	dmm	dmm	dmm	dmm	dmm	dmd	dmd
0.4	dmm	dmm	dmm	dmm	dmm	dmd	dmd
0.5	dmm	dmm	dmm	dmm	dmm	dmd	dmd
0.6	dmm	dmm	dmm	dmm	dmm	dmd	dmd
0.7	ddm	ddm	ddm	ddm	ddm	ddd	ddd
0.8	ddm	ddm	ddm	ddm	ddm	ddd	ddd

(a) Estimation results

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	dmm	dmm	dmm	dmm	dmm	dmm	dmd
0.3	dmm	dmm	dmm	dmm	dmm	dmd	dmd
0.4	dmm	dmm	dmm	dmm	dmd	dmd	dmd
0.5	dmm	dmm	dmm	dmm	ddm	ddm	dmd
0.6	dmm	dmm	dmm	dmm	ddm	ddd	ddd
0.7	ddm	ddm	ddm	ddm	ddm	ddd	ddd
0.8	ddm	ddm	ddm	ddm	ddm	ddd	ddd

(b) Experimental results

$\rho_2 \backslash \rho_3$	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.2	–	–	–	–	–	0.06	–
0.3	–	–	–	–	–	–	–
0.4	–	–	–	–	0.06	–	–
0.5	–	–	–	–	0.04	0.03	–
0.6	–	–	–	–	0.05	0.06	0.04
0.7	–	–	–	–	–	–	–
0.8	–	–	–	–	–	–	–

(c) Error rates

Figure 11: R8

[ObjectSpace Inc., 1997] ObjectSpace Inc. (1997). *JGL Version 3.1 User Guide*.

[ONTOS Inc., 1994] ONTOS Inc. (1994). *ONTOS Object Database Version 3.0 Developer's Guide*.

[Özsu and Blakeley, 1995] Özsu, M.T. and Blakeley, J.A. (1995). Query processing in object-oriented database systems. In Kim, W. editor, *Modern Database Systems : the object model, interoperability, and beyond*, pages 146–174. ACM Press/Addison-Wesley.

[Sauer and Chandy, 1981] Sauer, C.H. and Chandy, K.M. (1981). *Computer systems performance modeling*. Prentice-Hall, N.J.

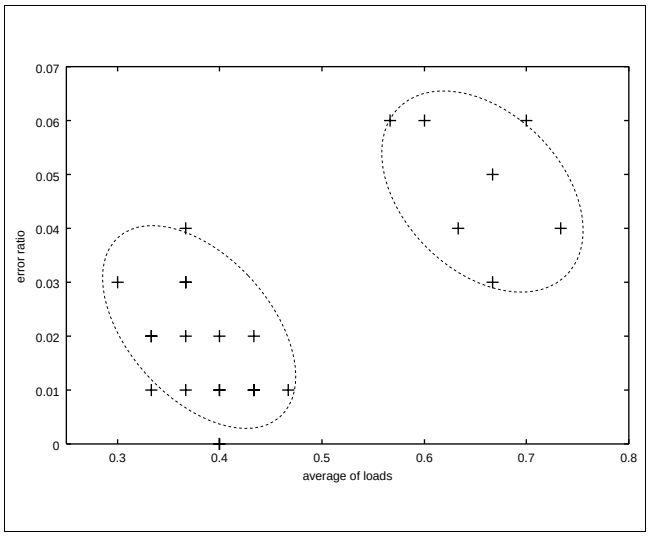

Figure 12: The correlation between average of load rates and error rate