

**GLOBAL MEMORY
MANAGEMENT
IN
CLIENT - SERVER SYSTEMS**

A THESIS

**SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE**

by

Yusemin TURKAN

June, 1995

**QA
76.9
.C55
T87
1995**

**GLOBAL MEMORY
MANAGEMENT
IN
CLIENT-SERVER SYSTEMS**

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by

Yasemin TURKAN

June, 1995

Yasemin Turkan
tasarimci ve yazilimci

Wh

76.9

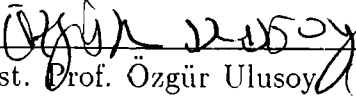
055

787

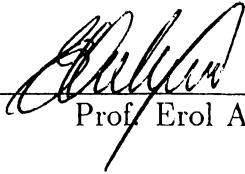
1995

B031258

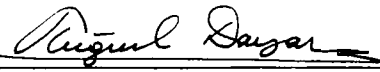
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Asst. Prof. Özgür Ulusoy (Advisor)

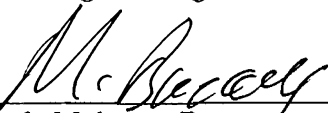
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Prof. Erol Arkun

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Asst. Prof. Tuğrul Dayar

Approved for the Institute of Engineering and Science:


Prof. Mehmet Baray
Director of the Institute

ABSTRACT

GLOBAL MEMORY MANAGEMENT IN CLIENT-SERVER SYSTEMS

Yasemin TURKAN

M.S. in Computer Engineering and Information Science

Advisor: Asst. Prof. Özgür Ulusoy

June, 1995

This thesis presents two techniques to improve the performance of the global memory management in client-server systems. The proposed memory management techniques, called “Dropping Sent Pages” and “Forwarding Sent Pages”, extend the previously proposed techniques called “Forwarding”, “Hate Hints”, and “Sending Dropped Pages”. The aim of all these techniques is to increase the portion of the database available in the global memory, and thus to reduce disk I/O. The performance of the proposed techniques is experimented using a basic page-server client-server simulation model. The results obtained under different workloads show that the memory management algorithm applying the proposed techniques can exhibit better performance than the algorithms that are based on previous methods.

Keywords: Client-Server Systems, Global Memory Management, Cache Consistency.

ÖZET

İSTEMCİ-SUNUCU SİSTEMLERDE GENEL BELLEK YÖNETİMİ

Yasemin TURKAN

Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans

Danışman: Asst. Prof. Özgür Ulusoy

Haziran, 1995

Bu tezde istemci-sunucu sistemlerin performansını arttırmak amacıyla iki teknik tanıtılmıştır. Önerilen teknikler, “Gönderilen Sayfaların Bellekten Atılması” ve “Bellekten Atılan Sayfaların Diğer İstemcilere Gönderilmesi”, daha önceden sunulmuş olan genel bellek yönetim tekniklerini (“Sayfa İsteklerinin Diğer İstemcilere Yönlendirilmesi”, “Gönderilen Sayfaların İşaretlenmesi” ve “Bellekten Atılan Sayfaların Sunucuya Geri Gönderilmesi”) geliştirmektedir.

Tezde önerilen tekniklerin performansı istemci-sunucu özelliklerini sağlayan bir benzetim modeli kullanılarak incelenmiştir. Değişik iş yükleri altında toplanan sonuçlardan, önerilen tekniklerin bellekte bulunan veritabanı büyüklüğünü arttırdığı gözlenmektedir. Böylece bu teknikler kullanılarak istemci-sunucu sistemlerde performans artışı sağlanmıştır.

Anahtar Sözcükler: İstemci-Sunucu Sistemler, Genel Bellek Yönetimini, Bellek Tutarlılığı.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor Asst. Prof. Özgür Ulusoy for his motivating support and endless help during my M.S. study. I would also like to thank Prof. Erol Arkun and Asst. Prof. Tuğrul Dayar for their invaluable comments on this thesis. I would not have managed to finish this study and reach my goal without unfailing support and help of my family. I would also like to thank my colleagues in ASELSAN and my friends in BILKENT for their great support. Finally, special thanks to my husband, Gökhan TURKAN, for his valuable comments, endless support and endless help throughout my studies.

Contents

1	Introduction	1
2	CLIENT-SERVER DATABASE SYSTEMS	3
2.1	The Query-Shipping Architecture	4
2.2	The Data-Shipping Architecture	4
2.2.1	The Object-Server Architecture	5
2.2.2	The Page-Server Architecture	6
2.2.3	Mixed Architectures	7
3	CACHE CONSISTENCY ALGORITHMS	8
3.1	Detection-based Protocols	8
3.2	Avoidance-based Protocols	10
3.2.1	Callback Locking Algorithm	11
4	DISTRIBUTED SHARED MEMORY SYSTEMS	13
4.1	Design Choices	14
4.2	Implementation Choices	17

4.2.1	Data location and access	17
4.2.2	Data Replication	18
4.2.3	Cache Consistency	19
4.2.4	Replacement strategy	19
5	GLOBAL MEMORY MANAGEMENT ALGORITHMS	21
5.1	Introduction	21
5.2	The Basic Algorithm	23
5.3	Franklin's Work	23
5.3.1	Forwarding	24
5.3.2	Hate Hints	24
5.3.3	Sending Dropped Pages	24
5.3.4	Forwarding with Hate Hints and Sending Dropped Pages Algorithm (FWD-HS)	25
5.4	The Proposed Memory Management Algorithm	26
5.4.1	Forwarding Dropped Pages	26
5.4.2	Dropping Sent Pages	26
5.4.3	Forwarding with Sending Dropped Pages, Forwarding Dropped Pages and Dropping Sent Pages Algorithm(FWD- SFD)	27
5.5	Performance Tradeoffs	31
6	THE CLIENT-SERVER DBMS SIMULATION MODEL	32
6.1	System Components	33

<i>CONTENTS</i>	viii
6.1.1 Client Model	33
6.1.2 Server Model	35
6.1.3 Network Model	35
6.2 Execution Model	36
6.2.1 CSIM: A C-Based, Process-Oriented Simulation Language	37
6.3 Database Model	38
6.4 Physical Resource Model	38
6.5 Workload Models	40
7 Performance Experiments	43
7.1 HOTCOLD Workload	45
7.1.1 Portion of Database Available in Memory	45
7.1.2 Resource Requirements	49
7.1.3 Throughput Results	52
7.2 PRIVATE Workload	58
7.3 UNIFORM Workload	66
7.4 HICON Workload	70
7.5 Summary of Results	74
8 CONCLUSION	76

List of Figures

4.1	Storage Hierarchy	14
5.1	How a client handles the page access request of a transaction.	27
5.2	Handling of a lock request issued by a client.	28
5.3	How the page access request of a client is handled by the server.	29
5.4	How a forwarded PageRequest is handled by a client.	30
6.1	Client-Server Architecture	33
6.2	Client Model	34
6.3	Server Model	35
7.1	% of DB Available in Memory (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	47
7.2	Total Disk I/O per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	47
7.3	Messages Sent per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	47
7.4	Message Volume per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	47

7.5	Server Buffer Hit Ratio (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	48
7.6	Server Misses Forwarded (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	48
7.7	Dropped Pages Kept in Memory per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	48
7.8	Dropped Pages per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	48
7.9	Throughput (HOTCOLD, 50% Ser Buf, 5% Cli Bufs, Slow Net).	54
7.10	Throughput, (HOTCOLD, 50% Ser Buf, 10% Cli Bufs, Slow Net).	54
7.11	Throughput (HOTCOLD, 50% Ser Buf, 25% Cli Bufs, Slow Net).	54
7.12	Throughput (HOTCOLD, 50% Ser Buf, 50% Cli Bufs, Slow Net).	54
7.13	Throughput (HOTCOLD, 50% Ser Buf, 5% Cli Bufs, Fast Net).	55
7.14	Throughput (HOTCOLD, 50% Ser Buf, 10% Cli Bufs, Fast Net).	55
7.15	Throughput (HOTCOLD, 50% Ser Buf, 25% Cli Bufs, Fast Net).	55
7.16	Throughput (HOTCOLD, 50% Ser Buf, 50% Cli Bufs, Fast Net).	55
7.17	Throughput (HOTCOLD, 10% Ser Buf, 10% Cli Bufs, Slow Net).	57
7.18	Throughput (HOTCOLD, 100% Ser Buf, 10% Cli Bufs, Slow Net).	57
7.19	Throughput (HOTCOLD, 10% Ser Buf, 10% Cli Bufs, Fast Net).	57
7.20	Throughput (HOTCOLD, 100% Ser Buf, 10% Cli Bufs, Fast Net).	57
7.21	Throughput (PRIVATE, 50% Ser Buf, 5% Cli Bufs, Slow Net).	60
7.22	Throughput (PRIVATE, 50% Ser Buf, 10% Cli Bufs, Slow Net).	60
7.23	Throughput (PRIVATE, 50% Ser Buf, 25% Cli Bufs, Slow Net).	60

7.24	Throughput (PRIVATE, 50% Ser Buf, 50% Cli Bufs, Slow Net).	60
7.25	Server Buffer Hit Ratio (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	61
7.26	Dropped Pages Kept in Memory per Commit (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	61
7.27	Message Volume per Commit (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	61
7.28	Dropped Pages per Commit (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).	61
7.29	Throughput (PRIVATE, 50% Ser Buf, 5% Cli Bufs, Fast Net).	64
7.30	Throughput (PRIVATE, 50% Ser Buf, 10% Cli Bufs, Fast Net).	64
7.31	Throughput (PRIVATE, 50% Ser Buf, 25% Cli Bufs, Fast Net).	64
7.32	Throughput (PRIVATE, 50% Ser Buf, 50% Cli Bufs, Fast Net).	64
7.33	Throughput (PRIVATE, 10% Ser Buf, 10% Cli Bufs, Slow Net).	65
7.34	Throughput (PRIVATE, 100% Ser Buf, 10% Cli Bufs, Slow Net).	65
7.35	Throughput (PRIVATE, 10% Ser Buf, 10% Cli Bufs, Fast Net).	65
7.36	Throughput (PRIVATE, 100% Ser Buf, 10% Cli Bufs, Fast Net).	65
7.37	Throughput (UNIFORM, 50% Ser Buf, 5% Cli Bufs, Slow Net).	67
7.38	Throughput (UNIFORM, 50% Ser Buf, 10% Cli Bufs, Slow Net).	67
7.39	Throughput (UNIFORM, 50% Ser Buf, 25% Cli Bufs, Slow Net).	67
7.40	Throughput (UNIFORM, 50% Ser Buf, 50% Cli Bufs, Slow Net).	67
7.41	Throughput (UNIFORM, 50% Ser Buf, 5% Cli Bufs, Fast Net).	68
7.42	Throughput (UNIFORM, 50% Ser Buf, 10% Cli Bufs, Fast Net).	68

7.43	Throughput (UNIFORM, 50% Ser Buf, 25% Cli Bufs, Fast Net).	68
7.44	Throughput (UNIFORM, 50% Ser Buf, 50% Cli Bufs, Fast Net).	68
7.45	Throughput (UNIFORM, 10% Ser Buf, 10% Cli Bufs, Slow Net).	69
7.46	Throughput (UNIFORM, 100% Ser Buf, 10% Cli Bufs, Slow Net).	69
7.47	Throughput (UNIFORM, 10% Ser Buf, 10% Cli Bufs, Fast Net).	69
7.48	Throughput (UNIFORM, 100% Ser Buf, 10% Cli Bufs, Fast Net).	69
7.49	Throughput (HICON, 50% Ser Buf, 5% Cli Bufs, Slow Net). . .	71
7.50	Throughput (HICON, 50% Ser Buf, 10% Cli Bufs, Slow Net).	71
7.51	Throughput (HICON, 50% Ser Buf, 25% Cli Bufs, Slow Net).	71
7.52	Throughput (HICON, 50% Ser Buf, 50% Cli Bufs, Slow Net).	71
7.53	Throughput (HICON, 50% Ser Buf, 5% Cli Bufs, Fast Net). . .	72
7.54	Throughput (HICON, 50% Ser Buf, 10% Cli Bufs, Fast Net). . .	72
7.55	Throughput (HICON, 50% Ser Buf, 25% Cli Bufs, Fast Net). . .	72
7.56	Throughput (HICON, 50% Ser Buf, 50% Cli Bufs, Fast Net). . .	72
7.57	Throughput (HICON, 10% Ser Buf, 10% Cli Bufs, Slow Net).	73
7.58	Throughput (HICON, 100% Ser Buf, 10% Cli Bufs, Slow Net). . .	73
7.59	Throughput (HICON, 10% Ser Buf, 10% Cli Bufs, Fast Net). . .	73
7.60	Throughput (HICON, 100% Ser Buf, 10% Cli Bufs, Fast Net). . .	73

List of Tables

5.1	Memory Hierarchy Costs	22
5.2	Memory Hierarchy Costs	23
6.1	Database Parameters	38
6.2	Physical Resource Parameters	39
6.3	Workload Parameters	40
6.4	Workload Parameter Settings for Client n	41
7.1	Simulation Parameter Settings	44

Chapter 1

Introduction

Recent improvements in computer hardware technology have increased the computing capabilities of the desktop machines. Therefore, within the last years, client-server systems have become more popular than the larger computer systems. Client-server systems provide access to the shared resources over a local communication network. In the past, there were no high performance workstations, and all the work was being done by a shared-server, through the use of SQL queries. Now the tendency is to move the computing functions from the servers to the clients. As the price/performance characteristics of the client-server systems are becoming very attractive, heavy research is being conducted in this area to improve the performance of these systems. Global memory management is one of the most active research areas in client-server systems. The global memory of a client-server system is composed of the server's buffer¹ and the clients' buffers. The methods developed for global memory management in client-server systems aim to reduce disk I/O by increasing the portion of the database available in the global memory.

Three global memory management techniques (Forwarding, Hate Hints, Sending Dropped Pages) are proposed by Franklin in [5]. These techniques provide, respectively, that: 1) clients utilize the entire memory of the system by obtaining pages from other clients, 2) page replacement policies at the server are modified to reduce replication of the pages between the server's and the

¹We use the terms "buffer" and "cache" interchangeably in this thesis.

clients' buffers, 3) client caches are extended by migrating pages from clients' buffer to the server's buffer.

In this thesis, we propose two additional global memory management techniques (Dropping Sent Pages, Forwarding Dropped Pages) for client-server systems. In these techniques: 1) a new extension is provided to the page replacement policies at the server to reduce the replication of pages in the global memory, and 2) the client caches are further extended by migrating pages from a client's buffer to not only the server's buffer, but also to the other clients' buffers. The performance of the proposed techniques against a base algorithm and the techniques proposed in [5] is examined under various workloads.

The next chapter describes the main architectural alternatives of the client-server database systems. Chapter 3 describes the main cache consistency algorithms and a global memory management algorithm that is used as the base algorithm in our experiments. As the client-server systems have adapted many characteristics of the distributed shared memory systems, Chapter 4 discusses the design and implementation choices of the distributed shared memory systems. The global memory management algorithms in client-server systems, and the techniques proposed to improve the performance of the algorithms are described in Chapter 5. Chapter 6 presents the client-server DBMS simulation model developed to test the proposed techniques. The performance results under different workloads are discussed in Chapter 7. Finally, Chapter 8 summarizes the conclusions of the thesis and suggests some possible extensions for the future work.

Chapter 2

CLIENT-SERVER DATABASE SYSTEMS

A client-server system is a distributed software architecture that executes on a network of interconnected desktop machines and shared server machines[5]. Client processes that interact with individual users, run on the desktop machines, where server processes are the shared resources that give services to the requesting clients. All the system is interconnected via a network. The partitioning responsibility that is inherent in the client-server structure accommodates the needs of users while providing support for the management, coordination, and control of the shared resources of the organization[5]. Therefore the client-server systems provide an efficient use of the shared resources in a distributed system.

Concerning the unit of interaction between client and server processes, two different architectural alternatives exist for the client-server database management systems: the query-shipping architecture and the data-shipping architecture. The following sections describe these architectural alternatives.

2.1 The Query-Shipping Architecture

In query-shipping systems, a client sends a query to the server; the server then processes the query and sends the result back to the client [5]. Most commercial relational database systems have adopted the query-shipping architecture, since it is more suitable for these systems. Query-shipping systems were very popular in the times where there were not many powerful desktop machines. In today's systems all the processing power is provided by big, powerful, and expensive mainframes, and the interaction with the individual users is provided using cheap and less powerful desktop machines. The interface between the server and the client is provided by a data manipulation language such as SQL. The advantages of the query shipping architecture can be listed as follows:

- Communication costs and client buffer space requirements are reduced since only the data items that satisfy a given query are transferred from the server to the clients[14]. This is especially important in a wide area network[18].
- Query-shipping provides a relatively easy migration path from an existing single-site system to the client-server environment since the database engine (which resides solely on the server) can have a process structure similar to that of a single-site database. Therefore, standard solutions for issues such as concurrency control and recovery can be used[5].
- Interaction at the query level facilitates interaction among heterogeneous database systems using a standardized query language such as SQL[5].
- The cost of clients is less, because no high performance desktop machine is needed for the data processing as all data processing is done on the servers.

2.2 The Data-Shipping Architecture

Data-shipping systems perform the bulk of the work of query processing at the clients, and as a result, much more DBMS functionality is placed at

the clients[5]. All commercial object oriented database management system (OODBMS) products and recent research prototypes use the data-shipping architecture. As the technology is advancing, the CPU, memory, and disk capacities of the desktop machines, and the network capacity and speed are increasing rapidly. Also, the prices of these high technology products are decreasing at a considerable rate. Therefore, the restrictions to use the data-shipping architecture are being removed. The advantages of the data-shipping architecture over the query-shipping architecture are[5]:

- The data-shipping approach offloads functionality from the server to the clients. This might be crucial for performance, as the majority of the processing power and memory in a workstation-server environment is likely to be at clients.
- Scalability is improved because usable resources are added to the system as more clients are added. This allows the system to grow incrementally.
- Responsiveness is improved by moving data closer to the application and allowing the programmatic interface of OODBMSs to directly access that data in the application's address space.

According to the unit of data that is moved between the server and the clients, the data-shipping architecture can be implemented in two different ways: the system can involve either an object-server, or a page-server. Mixed architectures that integrate the characteristics of these two system are also possible.

2.2.1 The Object-Server Architecture

The unit of data that is transferred between the client and the server is an object. The locking protocol is also implemented in object-level. Object servers provide many advantages, among which:

- They are very suitable for OODBMSs; i.e. they can be easily implemented on such systems.

- Both the server and the workstation are able to run methods. A method that selects a small subset of a large collection can execute on the server, avoiding the movement of the entire collection to the workstation[4].
- The design of the concurrency control subsystem is simplified as the server knows exactly which objects are accessed by each application[4, 5].
- Client buffer space can be used efficiently, since only requested objects are brought into client's buffer[5].
- Communication cost can be reduced by sending objects between clients and servers rather than entire pages if the locality of reference is poor with respect to the contents of pages[5].
- If the size of the objects changes during execution, this does not cause any problem, since the unit of data that is transferred is an object, not a fixed sized page.

2.2.2 The Page-Server Architecture

The unit of data that is transferred between the client and the server is a page. The locking is also performed in page-level. The advantages of the page-server architecture over the object-server architecture are:

- The page-server architecture places more functionality on the clients, which is where the majority of the processing power in a client-server environment is expected. Therefore the server can spend most of its CPU power to perform concurrency control and recovery[4, 5].
- Since entire pages are transferred intact between the client and the server, the overhead on the server is minimized[4].
- Effective clustering of objects into pages can result in fewer messages between clients and servers, especially for scan queries[5].
- The design of the overall system is simpler[5]. For example, page-oriented recovery techniques can be efficiently supported in a page-server environment. Also, in a page-server system, there is no need to support two query processors: one for objects, and one for pages.

2.2.3 Mixed Architectures

Various research have attempted to integrate the object-locking and the page locking architectures to improve the performance of the client-server systems. An example of such systems is the NFS file-server architecture[19]. It is a variation of the page-server design in which the client software uses a remote file service to read and write database pages directly[4]. Although the file-server architecture is slow, it has many advantages.

- It reduces the overhead placed on the server.
- Since NFS runs in the operating system kernel, using it to read and write the database provides that user-level context switches can be avoided completely. As a result, the rate at which data can be retrieved by a remote workstation is improved.

Another mixed architecture, presented in [12], supports object-level locking in a page-server context. The adaptive granularity approach based on that architecture uses page-level locking for most pages, but switches to object level locking when finer-grained sharing is demanding. The unit of data transferred between the server and the clients is a page. The adaptive protocol was shown to outperform the pure object-level locking and page-level locking protocols under most of the conditions tested.

There are some other new systems based on mixed architectures. SHORE (Scalable Heterogeneous Object Repository) is one example to these systems. It is a persistent object system currently being developed at the University of Wisconsin. SHORE integrates the concepts from object oriented database systems and file systems. From the file system world, it draws object naming services, support for lower (and cheaper) degrees of transaction-related services, and an object access mechanism for use by legacy UNIX file-based tools. From the object oriented database world, SHORE draws data modeling features and support for associative access and performance features.

Chapter 3

CACHE CONSISTENCY ALGORITHMS

Cache consistency maintenance is one of the main problems in client-server systems. There are many researches being done in this area. Since client-server systems consist of distributed processes, cache consistency algorithms that have been developed for distributed systems, can easily be implemented in client-server systems. As an example, different versions of the Distributed Two Phase Locking Algorithm have been proposed for distributed database systems[7], and the client-server systems have adapted many approaches used with these algorithms.

The traditional cache consistency algorithms are partitioned into two classes [5, 8, 9] : detection-based protocols and avoidance-based protocols. These protocols are described in the following subsections.

3.1 Detection-based Protocols

Detection-based protocols allow stale¹ data copies to reside in clients' cache. The server is responsible for maintaining the information that enables clients

¹If a copy of the data has been updated by another client, the copy residing in the client cache becomes invalid. This invalid copy is called the "stale copy".

to perform the validity checking. Transactions must check the validity of each cached page they access before they are allowed to commit. Transactions that have accessed stale data are not allowed to commit. Therefore consistency checks for the updated data should be completed during the execution of a transaction. The consistency action initiation can be done in three ways: [5].

- Synchronously, on each initial access to a page by a transaction.
- Asynchronously, on the initial access.
- Deferred, until a transaction enters its commit phase.

Stale data residing in caches increases the risk that a transaction will be forced to abort as a result of accessing that copy. Therefore, to reduce this risk, a change notification is sent to a remote client as a result of an update that could impact the validity of an item cached at the client [5]. Notifications can be sent either during the execution of the transaction or after the commit time. Remote updates after the notification can be done in three ways:

- Propagation : the stale copy in the remote client is updated by the new value.
- Invalidation : the stale copy in the remote client is removed from its cache.
- Dynamic : based on the current workload, the stale copy is either propagated or invalidated.

The Caching Two-Phase Locking protocol presented in [8, 21] is an example to detection-based protocols. In this protocol, the consistency action initiation is performed synchronously on each initial access, and no change notification hints are sent as a result of an update. Another example is No-Wait Locking protocol [21]. No-Wait Locking protocol works the same as the caching two-phase locking protocol, except that in this protocol the consistency action initiation is performed on initial access asynchronously. No-Wait Locking w/Notification[21] is an advanced version of the no-wait locking protocol, in which change notification hints are sent after commitment. The remote update

of the data is performed by propagation. There are other optimistic detection-based protocols, in which the consistency action initiation is deferred until commitment. The Optimistic w/Notification [6] is an example to these protocols. In this protocol, the change notification hints are sent after commitment, and the stale remote data is invalidated.

3.2 Avoidance-based Protocols

Under the avoidance-based protocols, transactions never have the opportunity to access stale data. Avoidance based protocols are based on the read one/write all (ROWA) replica management protocol. This protocol prevents transactions to access stale data in their local caches. Even though this protocol increases the complexity, it reduces the reliance on the server and thereby it is more suitable for the client-server architecture. What is more, transactions do not need to perform consistency maintenance operations during the commit phase, since intentions are obtained during transaction execution. In avoidance-based protocols the write intention declaration can be performed in three different ways[5]:

- Synchronously, on write intention fault.
- Asynchronously, on write intention fault.
- Deferred until commit.

In addition to the time write intentions are declared, avoidance-based algorithms can also be differentiated according to how long write intentions are retained for[5]. There are two alternatives:

- The write intentions can be retained for the duration of a particular transaction.
- The obtained write intentions can last until the clients want to drop it.

In avoidance-based protocols, there are two different remote conflict actions: wait and preempt. Under wait policy, if there exists any consistency action that conflicts with the operation of an ongoing transaction, it must wait for the transaction to complete. On the other hand, under a preempt policy, ongoing transactions can be aborted as a result of an incoming consistency action[5].

Similar to the detection-based protocols, remote updates can be implemented in three different ways: using invalidation, propagation, and dynamic.

Different versions of the Optimistic Two-Phase Locking Algorithm (O2PL) [8, 9] are examples to avoidance-based protocols. They all defer write intentions until commitment. The obtained write intentions last until the end of transaction, and they use wait policy for the remote conflict action. Versions of O2PL algorithms differ according to the remote update implementation. Among these versions are O2PL-Invalidate, O2PL-Propagate, O2PL-Dynamic, and O2PL-NewDynamic.

Notify Locks Algorithm [6], also defers the write intention declaration until the commitment, and the intention lasts until the end of transaction. However, under this algorithm preempt policy is used for the remote conflict action, and the remote updates are propagated.

Callback locking algorithms declare write intentions synchronously, and remote updates are implemented by invalidation. There are two different callback locking algorithms: Callback-Read[9, 21] and Callback-All[9]. In Callback-Read algorithm write intentions last until the end of the transaction, while in Callback-All write intentions are kept until they are revoked by the server or dropped by the client. Most of the client-server systems use the callback locking algorithm for the cache-consistency maintenance. Next subsection explains the callback locking algorithm.

3.2.1 Callback Locking Algorithm

In the callback locking a client must declare its intention to the server before granting any lock to a transaction, if the client does not have an update intention registered for the requested page. When a client declares its intention

to update a page of which other clients also have the copies of the same page, the server sends “call-back” messages for the conflicting copies. The write intention is registered at the server only once the server has determined that all conflicting copies have been successfully called back. The write intentions are not revoked at the end of a transaction. If a read request for a page arrives at the server and a write intention for the page is currently registered for some other clients, the server sends a downgrade request to those clients. A downgrade request is similar to a callback request, but rather than removing the page from its buffer, the client simply informs the server that it no longer has a registered write intention on the page. At a remote client, a downgrade request for a page copy must first obtain a read lock on the page. If a conflict arises, the downgrade request blocks, and a message is sent to the server informing the conflict.

Chapter 4

DISTRIBUTED SHARED MEMORY SYSTEMS

Hardware technology develops with a high speed. The CPU power of the workstations, the access time of the memories improve rapidly. However, memories are still faster than disks, and disks are still faster than tapes. As the access time of a storage medium increases, the cost of purchasing this medium also increases. Figure 4.1 shows a hierarchy of the storage devices. [15].

Main memory has the smallest access time, while it has the most cost value. The next levels in the hierarchy are formed by the extended memory, solid state disk, and disk cache. The bottom levels consist of magnetic disk, disk arrays, magnetic tape, and optical disk, which have maximum access time, but minimum cost. Different memory management algorithms are proposed to increase the access time, while maintaining low cost systems. Distributed shared memory systems are the examples to these system.

Distributed shared memory systems implement the shared memory abstraction on multicomputer architecture, combining the scalability of network based architecture with convenience of shared-memory systems [1]. Distributed shared memory research goals and issues are similar to those of research in multiprocessor caches or network file systems, memories for nonuniform memory access multiprocessors, and management systems for distributed or replicated databases [1]. Since client-server systems also consist of distributed processes,

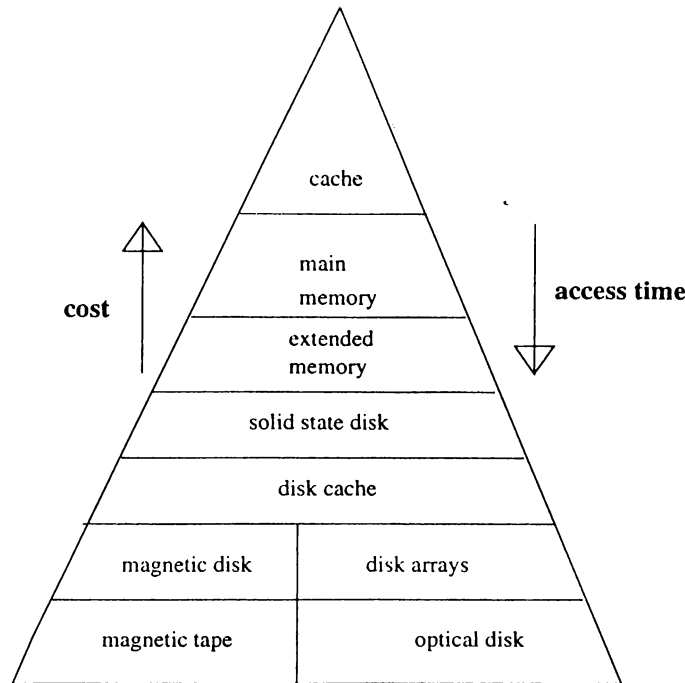


Figure 4.1. Storage Hierarchy

they have adapted many approaches used with these systems.

This chapter explains the design and implementation choices of distributed shared memory systems[1]. These choices can also be applied to client-server systems.

4.1 Design Choices

Distributed shared memory systems can be designed in different ways according to the choices of the network, the structure and the granularity of data, the cache consistency algorithm used, the scalability, and the heterogeneity of the system.

Network

The systems can be designed on different network choices, which can be:

- Common networks of workstations or minicomputers,
- Special-purpose message-passing machines (such as Intel iPC/2),
- Custom hardware,
- Heterogeneous systems.

Structure and Granularity

Structure refers to the layout of the shared data in memory[1]. The layout can be in two different ways:

- Unstructured memory: a linear array of words.
- Structured memory: objects, language types, etc.

Granularity refers to the size of the unit of sharing: byte, word, page or complex data structure. Hardware implementations of distributed shared systems support smaller grain sizes like byte, word, etc.[1]. When a complex data is used for the size of the sharing, the size of the granularity changes according to the application. If the size of unit of sharing is page, there are tradeoffs between the larger and the smaller page sizes. Having a larger page size has the following advantages over having smaller page size:

- When there is locality of reference, having a larger page size increases the performance, and reduces the overhead.
- When there is a need to keep directory information about pages in the system, larger pages reduce the size of the directory.

Having a smaller page size has the following advantages over having a larger page size:

- Having a smaller page size reduces the probability of sharing the same page. This reduces the data contention.
- Smaller page sizes decrease the risk of false sharing that occurs when two unrelated data are placed in the same page.

Cache Consistency

The data consistency in caches can be obtained with the following consistency choices[1]:

- **Strict consistency:** A read returns the most recently written value.
- **Sequential consistency:** The result of any execution appears as some interleaving of the operations of the individual nodes when executed on a multithreaded sequential machine.
- **Processor consistency:** Writes issued by each individual node are never seen out of order, but the order of writes from two different nodes can be observed differently.
- **Weak consistency:** The programmer enforces consistency using synchronization operators guaranteed to be sequentially consistent.
- **Release consistency:** Weak consistency with two types of synchronization operators: acquire and release. Each type of operator is guaranteed to be processor consistent.

Even though the strict consistency algorithms reduce ambiguity, relaxed consistency algorithms increase the performance of the systems, as less synchronization messages are sent and less data movements are implemented.

Scalability

Distributed shared memory systems scale better than tightly coupled shared-memory multiprocessors[1]. Mainly two factors limit the scalability:

- Central bottlenecks (i.e., network connecting processors, processors themselves, etc.).
- Global knowledge operations and storage (i.e., broadcast messages, directory information to keep track of the location of the pages, etc.).

Heterogeneity

Heterogeneity is having different machines sharing the same memory. Different algorithms can run on different machines. Data has to be converted when it is passed between two different nodes, and this increases the overhead. The performance of the system decreases.

4.2 Implementation Choices

In addition to the design choices in distributed shared memory systems, there are different implementation choices. They are stated as follows:

4.2.1 Data location and access

Data is either stored in a stated location or it is migrated throughout the system[13]. It is easier to keep track of data if it is located, in a stated site. Some systems distribute the data among nodes with a hashing algorithm. Although these system are simple and fast, they may cause bottleneck under the condition that the data is not distributed properly. If the data is migrated throughout the system, there are different ways of memory management:

- The server keeps track of the location of the data. Even though it is easier to implement, the server becomes a bottleneck.
- Broadcast requests can be sent to all the sites for the required data. Sending broadcast messages increases the communication of the system and the network becomes a bottleneck. Different algorithms using broadcast messages are proposed in [3]. The algorithms apply dynamic policies for remote caching.
 - Passive-sender/Passive-receiver Algorithm: The strategy aims to minimize the amount of communication. The sender does not actively hand over any data. If it needs to replace any data, it just

broadcasts it to the network. If any server is listening, the data might be picked up.

- Active-sender/Passive-receiver Algorithm: A workstation trying to get rid of some valuable data, takes the initiation to hand over the data to another workstation. It sends a broadcast message asking for an idle or a less loaded workstation. From the fast responding workstations, the sender chooses one and hands over the data.
 - Passive-sender/Active-receiver Algorithm: If a workstation becomes idle, it takes the initiative to obtain globally valued data from the data server or an overflowing workstation. When a workstation becomes idle, it sends a broadcast message telling that it is idle to receive data. Upon receiving this message other workstations start sending dropped data to the idle workstation.
 - Active-sender/Active-receiver Algorithm: This strategy combines the active roles of sender and receiver, possibly combining the benefits of both. When a node is idle, it volunteers to store the data of other nodes. When it becomes a bottleneck, it asks other nodes to store its data.
- Another memory management technique proposed to handle data migration is called owner-based distributed. Each piece of data has an associate owner; i.e., a node with the primary copy. The owners change as the data migrates through the system[1]. If another node requests the data from the owner, the owner checks for the data in its cache. If the node has the data, it sends the data to the requesting client. Otherwise, it forwards the request to the node that it had sent the data previously. If the data is migrated many times from one node to another, the message also follows this path and the communication increases, so that the network becomes a bottleneck.

4.2.2 Data Replication

If the data is replicated, the replication protocol can be implemented in two different ways[13]:

- Read-replication: Copies of the data exist in the system for read access. However, if a write access is to be performed on the data, all the other copies are invalidated.
- Full-replication: Copies of the data exist in the system for both read and write accesses. Even when a write access is to be performed on the data, other copies are not invalidated, and the data is replicated.

4.2.3 Cache Consistency

If the shared data is not replicated, then the cache consistency protocol is trivial. The requests that come via network, are automatically serialized by the network. If the shared data is replicated, a cache consistency algorithm must be applied to serialize requests. Cache consistency algorithms are discussed in Chapter 3.

4.2.4 Replacement strategy

In systems that allow data migration, the available space for caching data may fill up. Under this condition, replacement strategy determines which data should be replaced to free space and where the replaced data should go. Two different algorithms are used to determine which data should be replaced:

- Least Recently Used(LRU) data can be replaced.
- Values can be given to each data, and the least valuable ones can be replaced [3].

Three different algorithms are used to determine where the replaced data should go:

- Replaced data is just dropped out.
- Valuable replaced data are kept by other remote workstations in the system[2, 3].

- If the replaced data is the only copy in the system, it is transferred to the home node [1, 5].

Chapter 5

GLOBAL MEMORY MANAGEMENT ALGORITHMS

The global memory management algorithm that we have developed is described in this chapter. The first section makes a brief introduction to the global memory management in client-server systems. The following sections describe a basic memory management algorithm, Franklin's extension to the basic algorithm[5], and finally the algorithm that we propose.

5.1 Introduction

The client-server systems have adapted many of the design and implementation choices of the distributed shared memory systems discussed in Chapter 4. The client-server systems in general, consist of the following components:

- Clients: having main memory to cache a small part of the database for faster access time.
- Server: having slow disks that hold the whole database, and fast main memory that holds a large portion of the database.

- Network: to connect clients and the server to handle the communication among them.

The global memory management algorithms apply different choices in the memory hierarchy costs. The simplest algorithm works as follows: If a client needs a database page, it first searches its local cache. Under the condition that a miss occurs, it requests the page from the server. The server searches its main memory for the page. If it finds the page in its memory, it sends the page to the client. Otherwise, it reads the page from its disk to its memory, and then sends the page to the client. Data messages are large messages relative to other kinds of messages. The global memory hierarchy costs in this algorithm are shown in Table 5.1.

Rank	Level	Small Messages	Large Messages	Disk I/Os
1	Local Memory	0	0	0
2	Server Memory	1	1	0
3	Server Disk	1	1	1 or 2

Table 5.1. Memory Hierarchy Costs

Another algorithm that exploits the contents of remote client memory, results in a four-level hierarchy[5] as shown in Table 5.2. If the requested page does not reside in the server's memory, the server looks for a remote client which has the page. If it finds such a client, it forwards the page request to the remote client, so that the remote client sends the page to the requesting client. If the page is not found in any of the clients, then the server reads the page from its disk to its memory.

The local memory access has the minimum access time and cost, while the server disk access has the maximum access time and cost, in terms of the storage hierarchy and the number of messages sent. Therefore, the aim of the global memory management algorithms is to move accesses from the highest ranks to lower ranks as stated in Table 5.2.

Rank	Level	Small Messages	Large Messages	Disk I/Os
1	Local Memory	0	0	0
2	Server Memory	1	1	0
3	Remote Memory	2	1	0
4	Server Disk	1	1	1 or 2

Table 5.2. Memory Hierarchy Costs

5.2 The Basic Algorithm

The basic algorithm that is used for performance analysis is the Callback-All(CB-A) Algorithm. This algorithm has been described in Section 3.2.1. Recall that in CB-A, the server must keep track of the page copies that are sent to the clients. The server is also responsible for maintaining the cache-consistency. It keeps track of the locks that each client obtains for a database page. If there exists a lock conflict, the server calls back the locks from the clients that owns a conflicting lock on the page. Before accessing a page clients must obtain the specified READ or WRITE lock on that page. Dirty pages of the committed transactions are copied back to the server. Both the server and the clients use the LRU algorithm for the replacement of the pages in their buffer.

5.3 Franklin's Work

Three global memory techniques, called Forwarding, Hate Hints and Sending Dropped Pages, are presented in [5, 11]. These techniques are originally based on the CB-R algorithm (see Section 3.2.1). However in our work, they are modified to be based on the CB-A algorithm that is explained in the previous section. The following subsections describe these techniques and the algorithm that has been developed by Franklin.

5.3.1 Forwarding

The major aim of this technique is to minimize disk accesses by extending the global memory. If a client requests a page from the server, the server first searches the page in its cache. If the page does not exist in the server's cache, the page request message is forwarded to the client that has a local copy of the page. Upon receipt of a forwarded request, the remote client sends the copy of the page to the requesting client. Therefore, instead of having disk I/O, a small message is sent from the server to the client who has a local copy of the page. This technique has the highest effect for the performance improvement.

5.3.2 Hate Hints

This technique is used to keep a larger portion of the database available in the global memory, when the forwarding technique is used. If the server sends a page to a client, it marks the page as hated¹. The page will be probably replaced, when an empty buffer frame is needed for a new page. Therefore, this technique reduces the replications of the pages in the global memory. When a page is sent to a client, it is known that the page is in the global memory.

5.3.3 Sending Dropped Pages

This technique can also be used to increase the portion of the database in the global memory, when forwarding technique is used. With this technique the server buffer pool is used to prevent a page to be completely dropped out of the global memory. If a client is going to drop a page, it informs the server by piggybacking this information to the page request message. If the page is the only local copy in the global memory, the server tells the client to send the dropped page, by piggybacking this information to the requested page. Otherwise, the page is dropped, because it has another copy that resides in the global memory.

¹i.e., it marks page as the least recently used one.

5.3.4 Forwarding with Hate Hints and Sending Dropped Pages Algorithm (FWD-HS)

The memory management algorithm proposed by Franklin[5] combines the three techniques described in the previous subsections. If a client needs a page that is not in its memory, it requests the page from the server. Upon receiving a page request, the server checks its buffer. If the page is in its buffer, the server sends the page to the client and marks the page as hated. If the page is not in the server's buffer, then the server searches for a remote client that has a local copy of the page. If such a client exists, the page request message is forwarded to that client. Upon receiving the forwarded request, the client sends the page to the client that has requested the page. Since the locks are obtained before page request messages are sent, the client can directly send the page to the requesting client.

There may not exist any local copy of the requested page in the global memory. In this case, the server reads the page from its hard disk, and sends the page to the client. Then the server marks that sent page as hated. Therefore, the page will be probably replaced, when a free space in the server buffer is required.

The buffer pool of a client can be filled up during the execution of a transaction. If the client has to replace a page, it sends this information by piggy-backing to the page request message. The server keeps track of the location of the page copies that reside in the global memory. If the page that is going to be dropped is not the only copy in the global memory, the server tells the client to directly drop the page from its buffer. Otherwise, the client is requested to send the page to the server. The server puts the page into its buffer, and marks the page as the most recently used (MRU) page, so that it is not dropped when an empty buffer location is needed. If the server's buffer is full, the server uses a page replacement policy (i.e., LRU) to receive the dropped page.

5.4 The Proposed Memory Management Algorithm

The algorithm that we have developed extends the Franklin's algorithm (FWD-HS). In addition to the three global memory management techniques used in FWD-HS, we propose two more techniques to improve the performance of the algorithm. We call these techniques Forwarding Dropped Pages and Dropping Sent Pages.

5.4.1 Forwarding Dropped Pages

This technique is used together with the Sending Dropped Pages and Forwarding techniques. It aims to increase the portion of the database in the global memory, and to reduce the disk I/O. When a client is going to drop a page that is the only copy in the global memory, the server tries to keep it in its memory. However, if its buffer is full, the server finds a remote client that has a free buffer space to receive the dropped page. Therefore, the only copy of the page is not dropped out of the global memory and no disk I/O is performed by the server to store that page into the disk.

5.4.2 Dropping Sent Pages

This technique extends the Hate Hints technique. After sending a page to a client, the server just drops the page from its memory instead of marking the page as hated and waiting it to be replaced. As the sent page is not the only copy in the global memory, this technique frees the server's buffer to be able to store more pages. Therefore, the portion of the database available in the global memory is increased when this technique is used with the Forwarding and the Forwarding Dropped Pages techniques.

As explained above, the proposed two techniques increase the portion of the database in the global memory. They are also expected to improve the performance of the other global memory management algorithms when the cost of

disk I/O is greater than the cost of transmitting and processing messages. The memory management algorithm proposed on the basis of these two techniques is explained in more detail in the following subsection.

```

if the required READ or WRITE lock was not obtained before
    Send LockRequest message to the server
    Wait until the lock is granted
end if // The required lock is obtained
if the requested page (Page(n)) is not in the buffer
    Search for free space in the buffer to place the page
    If no free space exists in the buffer
        Choose a Page(m) with the LRU value
        Piggyback this information (that Page(m) will be dropped)
        to PageRequest message
    end if
    Send PageRequest message to the server
    Receive Page(n)
    if Page(m) is going to be dropped
        Obtain the piggyback information about Page(m)
        According to this information if the option is
            Drop_Page : drop the page
            Send_Page_to_Server : send the page to the server
            Send_Page_to_Client(x) : send the page to the client x
    end if
end if //The page is brought into the client's buffer

```

Figure 5.1. How a client handles the page access request of a transaction.

5.4.3 Forwarding with Sending Dropped Pages, Forwarding Dropped Pages and Dropping Sent Pages Algorithm(FWD-SFD)

FWD-SFD algorithm basically uses the Forwarding Dropped Pages and Dropping Sent Pages techniques in addition to those defined in FWD-HS algorithm.

Figure 5.1 provides a procedural description of how a page access request of a transaction is handled at a client site. When a transaction executing at client wants to access a page, it must first obtain READ or WRITE lock. If it has already obtained the specified lock, it continues its execution. Otherwise, the client requests the lock from the server. The execution of the transaction is blocked until the lock is granted by the server.

```

Check for conflicting locks
if there exists any
  for all of the conflicting locks
    send callback message to the client that has the lock
    wait for the lock to be downgraded
  end for
end if
Send the lock to the requesting client

```

Figure 5.2. Handling of a lock request issued by a client.

The server keeps track of the locks obtained by the clients on all database pages. When a lock request is received, if no conflicting lock exists in the system, the server immediately grants the lock to the requesting client. However, if there exists any conflicting locks, the server sends a callback message to each of the clients that have a conflicting lock. If a client is not using the lock for its transaction, it downgrades the lock (see Section 3.2.1). Otherwise, it waits until the commit time to downgrade the lock. The server waits until all the conflicting locks are called back. Then it grants the lock to the client. The algorithmic description of this procedure is provided in Figure 5.2.

After obtaining the lock, the client can access the page in its buffer. If the page does not reside in the client's buffer, the client has to request the page from the server. Before sending the page request, the client checks its buffersize. If the client has to drop a page in order to receive a new one, it piggybacks this information to the PageRequest message that is going to be sent to the server.

Upon receiving a PageRequest message, the server searches its buffer. If

```
initialize ForwardPage and SendPage to FALSE
if the requested page (i.e., Page(n)) is not in the server's buffer
  if the page is in any of the other clients' (i.e., Client(y)'s) memory
    set ForwardPage to TRUE
  else //the page is not in the global memory
    read Page(n) from the disk
    set SendPage to TRUE
  end if
else //the page is in the server's buffer
  set SendPage to TRUE
end if
if there is a piggybacked drop information in the request message
  if the page that will be dropped (i.e., Page(m)) is not
    the only copy in the global memory
    set DropInfo to Drop-Page
  else //Page(m) is the only copy in the global memory
    if the server's buffer is not full
      set DropInfo to Send-Page-To-Server
    else if there is a Client(x) available to take the page
      set DropInfo to Send-Page-To-Client(x)
    else
      set DropInfo to Send-Page-To-Server
    end if
  end if
end if
if SendPage is TRUE
  if there is a drop information
    Piggyback DropInfo to Page(n)
  end if
  send Page(n) to the client
  remove Page(n) from the buffer
else // ForwardPage is TRUE
  if there is a drop information
    Piggyback DropInfo to the PageRequest message that is going to be forwarded
  end if
  forward PageRequest message to Client(y)
end if
if DropInfo is Send-Page-To-Server
  Receive dropped Page(m)
  if the buffer is full
    replace the LRU page in the buffer
  end if
  Mark the page as MRU
end if
```

Figure 5.3. How the page access request of a client is handled by the server.

the page is in its buffer, the server directly sends the page to the client and drops that page out of its buffer. If a copy of the page does not exist in the server's buffer, but resides in the global memory (in one of the clients' cache), the server forwards the PageRequest message to the remote client that has a local copy of the page. Under the condition that no copy of the page resides in the global memory, the server reads the page from its disk, and sends it to the client. In the FWD-HS algorithm, the page that is sent to a client by the server is marked as the LRU. The FWD-SFD algorithm modifies this technique in the sense that the server drops the sent pages out of its buffer.

When there is a piggyback information in the PageRequest message, stating that the client has to drop a page in order to receive a new one, the server checks if that page is the only copy in the global memory. If so, the page should not be dropped from the global memory. In the FWD-HS algorithm, the server tells the client to send the page directly to itself. If the server's buffer is full, the server replaces the least recently used page with the dropped page. However, in the FWD-SFD algorithm, the server tells the client to send the page to itself only if its buffer has an empty space to receive the page. Otherwise, the server searches for a client that can take the dropped page. If there exists any remote client that can store the page in its buffer, the server tells the client to send the page to that remote client. However, when the server can not find any remote client to send the dropped page, it tells the client to send the page to itself. The server then replaces that page with the one that has the LRU value. The procedure presented in Figures 5.3 describes how a page access is handled by the server.

```

if there is DropInfo piggybacked to the forwarded PageRequest message
    Piggyback the DropInfo to the requested Page(n)
end if
send Page(n) to the requesting client

```

Figure 5.4. How a forwarded PageRequest is handled by a client.

When the requested page is directly sent by the server, the drop information is piggybacked to the sent page. However, when the PageRequest message is forwarded, this information is piggybacked to the forwarded message, so that

the client which receives the forwarded message can piggyback this information to the page that is sent to the requesting client. This procedure is provided in Figure 5.4

The client receives the page it needs from either the server or any of the clients that has a local copy of the page. There might be a drop information piggybacked to the received page. Based on this information, the client either drops the page specified, or sends the page to the server, or it sends the page to another client.

Clients can receive dropped pages asynchronously from other clients. They insert the dropped pages into their buffers and mark them as the MRU pages.

5.5 Performance Tradeoffs

The previous sections describe the FWD-HS algorithm that extends the base algorithm CB-A, and the FWD-SFD algorithm that extends the FWD-HS algorithm. The CB-A algorithm does not exploit remote client memory, and it relies on the local client memory, server memory, and the disk. The FWD-HS algorithm extends the global memory, and reduces the disk I/O by using more server CPU and messages. The major technique that affects the performance is the forwarding technique [5, 11]. This algorithm increases the portion of the database in the global memory. Therefore page hit ratio of the global memory is increased. As the global memory hit ratio increases, disk I/O reduces.

The proposed algorithm FWD-SFD extends the FWD-HS algorithm. By using the Forwarding Dropped Pages and Dropping Sent Pages techniques, the algorithm increases the portion of the database in the global memory. Since these techniques tend to use less disk I/O, the FWD-SFD algorithm is expected to provide better performance than the other algorithms.

Chapter 6

THE CLIENT-SERVER DBMS SIMULATION MODEL

A client-server DBMS simulation model has been involved in our performance experiments. The simulation model is based on the client-server DBMS model presented in [20], which was heavily influenced by the client-server architecture described in [5]. The model in [20] has been extended to simulate the algorithms discussed in the previous chapter.

The client-server DBMS model simulates a data-shipping page server, where a number of clients are interacting via a local area network component. Clients generate transactions and request pages to complete the execution of their transactions. Their workload is derived from these transactions. Server's workload is generated by the requests coming from the clients. Concurrent transaction execution is not allowed on client workstations.

The global memory hierarchy of the system consists of client buffers, server buffer and server disks (where the database resides). Clients obtain page locks from the server. All concurrency control and consistency maintenance is implemented at a page granularity. Database pages with corresponding locks are cached in clients' buffers. Concurrency control, consistency maintenance, and buffer management are modeled in full detail, while database, network, and workload are modeled more abstractly. The general structure of the client-server architecture consisting of a single page server is depicted in Figure 6.1.

The following sections describe the simulation model in detail.

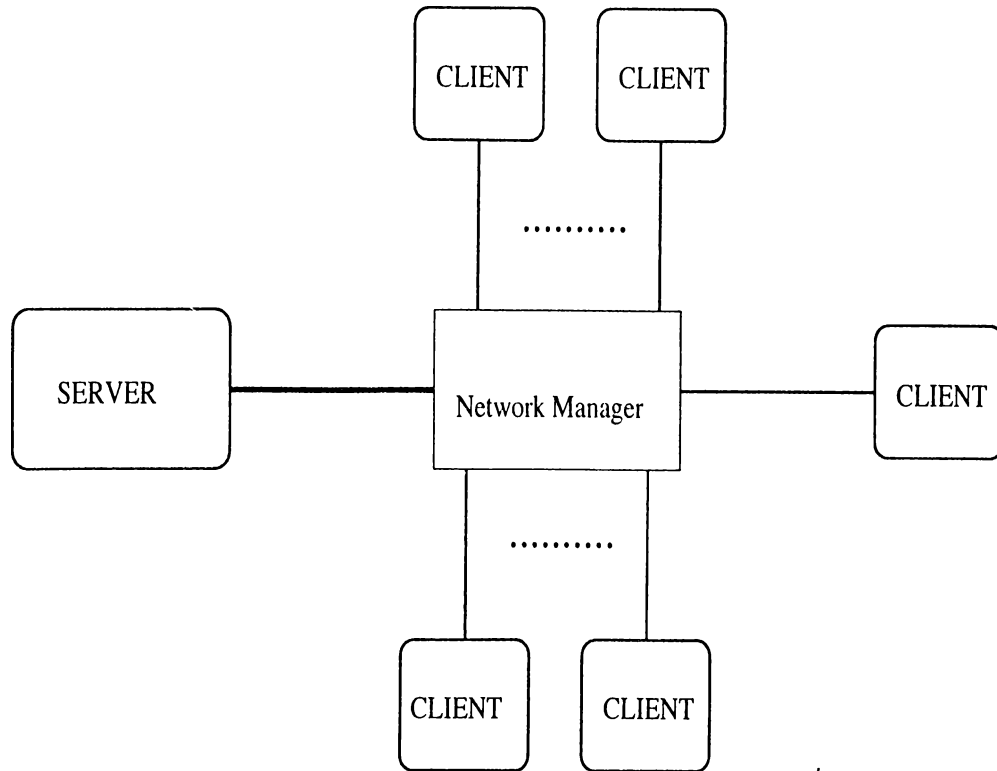


Figure 6.1. Client-Server Architecture

6.1 System Components

The basic components of the system are the client model, the server model, and the network model. Clients are connected to the server via a local area network. The following subsections describe the basic components that make up the simulation model.

6.1.1 Client Model

Each client consists of several modules as depicted in Figure 6.2.

The modules are presented as follows:

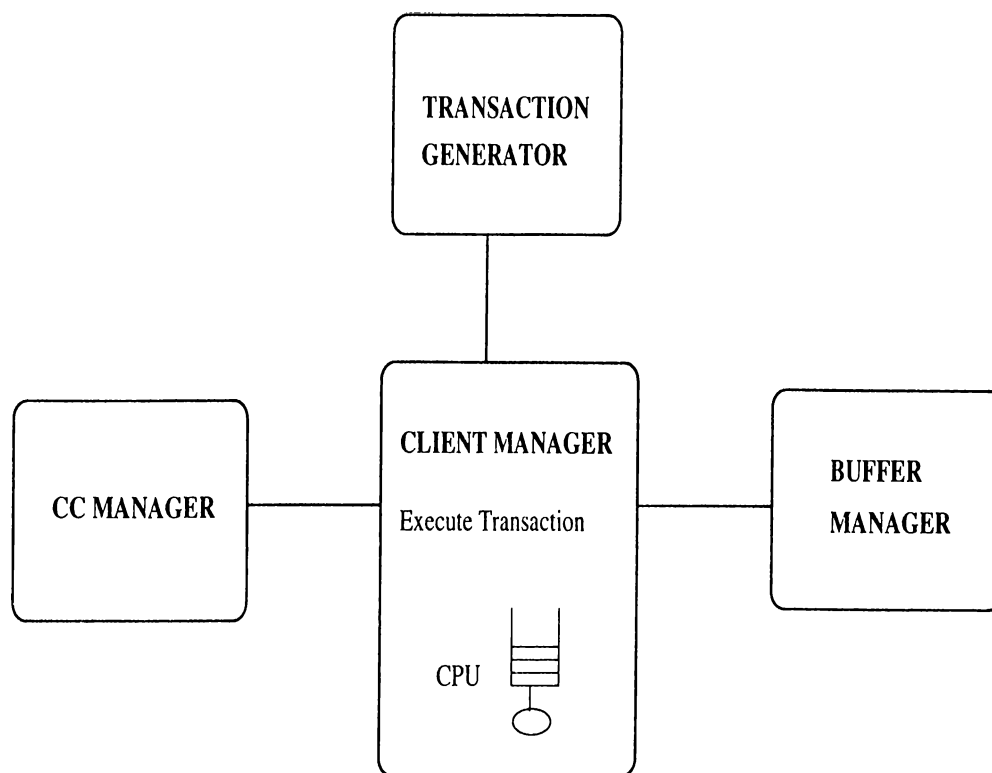


Figure 6.2. Client Model

- **Transaction Generator**: Generates transactions according to the workload model.
- **Client Manager** : Coordinates the operation of a client; executes transactions generated by the transaction generator, responds to messages sent by the server, processes requests received from other clients and the server.
- **Concurrency Control Manager** : Performs the concurrency-control operations in client, according to the specified cache consistency protocol. It is responsible for locking, unlocking and caching of pages.
- **Buffer Manager**: Manages the client buffer pool by using a specified page replacement policy.

6.1.2 Server Model

The server model, depicted in Figure 6.3, is similar to the client model.

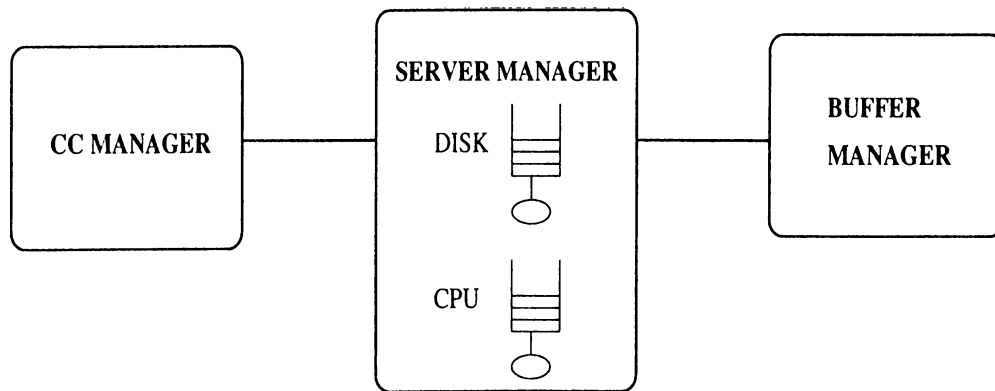


Figure 6.3. Server Model

The server model consists of the following modules:

- **Server Manager:** Coordinates the operation of the system. Handles the data requests coming from the clients.
- **Concurrency Control Manager:** Stores information about the location of page copies in the system, manages data replications, coordinates the sharing of locks.
- **Buffer Manager:** Manages the server buffer using a specified page replacement policy.

6.1.3 Network Model

The network manager is a simple module which simulates a LAN connecting the clients and the server. The network is modeled as a FIFO server with a fixed service rate. A failure-free network is assumed, so that there is no message losses.

6.2 Execution Model

The transaction generator module in each client, generates a stream of reference strings, that specify the transactions to execute. The client manager module executes the transactions in reference strings, according to the cache consistency algorithm Callback-All Locking (see Section 3.2.1). Before each access to a database page, the concurrency control(CC) manager is called by the client manager, The CC manager is responsible for obtaining the specified lock. If the client does not have the lock, the CC manager blocks the execution of the transaction and tries to obtain the lock from the server. While waiting for a lock from the server, the server may send an abort message for the transaction due to the deadlock detection mechanism used in the system. In that case, the client aborts the transaction, removes the locks obtained so far by the transaction, invalidates the pages updated by the transaction, and restarts the transaction. If, on the other hand, the lock is obtained, the client manager calls the buffer manager, which is responsible for the organization of the client's cache. If the page does not exist in the cache, the buffer manager blocks the transaction and requests the page from the server. If the buffer is full, the buffer manager uses a page replacement policy, to determine the page to be replaced. At commit time, the client sends a commit message to the server along with the list of pages updated by the transaction.

The workload of the server is determined by the requests coming from the clients. The server controls the whole system. It keeps track of the information necessary for the concurrency control management, using the Callback-All Locking algorithm. The server handles the requests of the clients in a FIFO order. The server grants a lock when all conflicting locks on the requested page are released. After granting the appropriate lock to the requesting client, the server sends the page to the client according to the specified global memory management algorithm.

All the messages are sent over a LAN. The network manager module simulates the message exchanges over the LAN among clients and the server. The messages are sent in a FIFO order.

Deadlocks are dealt by detection and recovery. One of the transactions involved in a deadlock is aborted. Deadlock detection frequency indicates the

frequency by which the server executes the global deadlock detection algorithm. In order to detect a deadlock a wait-for-graph is formed. If a deadlock is detected, one of the client transactions involved in the deadlock is aborted to break the deadlock cycle.

The simulation program consists of approximately 5,000 lines of code and it has been written in C, using CSIM: A C-Based, Process-Oriented Simulation Language [16, 17]. Usage of CSIM has greatly reduced the amount of code to be written. CSIM is explained briefly in the following subsection.

6.2.1 CSIM: A C-Based, Process-Oriented Simulation Language

CSIM is a process-oriented simulation language which is implemented over the C programming language. Therefore, programs written in CSIM are actually C programs that call C procedures. This allows the user to have all the power of C, with the additional features of creating process-oriented simulation models. The simulation programmers can easily define, initiate, and synchronize processes. In addition to these features, message passing, data collection and debugging can easily be implemented using CSIM.

The programmer can define a process that use system resources in one of three states:

- Active; i.e., currently being processed by the simulator.
- Holding; i.e., waiting for an interval of simulation time to pass.
- Waiting; i.e., waiting for an event to occur in a queue.

A system model can be defined by interacting processes as abstractions of the active entities in the system. All active processes can be executed in a pseudo-parallel fashion. CSIM takes care of the consistent execution of these processes, since the program is actually executing on a single processor.

Each process has a private data area, and active processes are able to communicate and synchronize with other active processes.

In the simulation model that has been developed, the three active entities of the system (i.e., the client module, the server module, and the network module) are defined as CSIM processes. These processes are then executed in a pseudo-parallel fashion. The mailbox facility of CSIM is used to exchange messages among processes. Random number distribution functions of CSIM (e.g., exponential, uniform, etc.) are used to generate the several workload sets.

6.3 Database Model

The database in our system consists of *DatabaseSize* number of pages of *PageSize* bytes each. As the model simulates a data-shipping page server, the unit of transfer and the granularity of locking is a single page. Parameters used to specify the database system are listed in Table 6.1

PARAMETER	MEANING
<i>DatabaseSize</i>	Size of the database in pages
<i>PageSize</i>	Page size in bytes

Table 6.1. Database Parameters

6.4 Physical Resource Model

The client and server instruction execution rates are specified by *ClientMIP* and *ServerMIP*, respectively. The FIFO policy is used in scheduling client and server CPU. Buffer sizes for each client and the server are given by *ClientBufSize* and *ServerBufSize*, respectively, and the buffers are managed using the specified page replacement policy. The server has a disk, and the disk has an access time that is uniformly distributed over the range from *MinDiskTime* to *MaxDiskTime*. Disk accesses are scheduled on a FIFO-basis. The server CPU processes *DiskOverheadInst* instructions for each I/O operation.

The network model uses a first-come-first-served policy in scheduling the

PARAMETER	MEANING
<i>ClientBufSize</i>	Client buffer size in pages
<i>ClientMip</i>	Instruction execution rate of each client (in MIPs)
<i>ControlMsgSize</i>	Control message size in bytes
<i>DeadlockInterval</i>	Time interval between two successive deadlock detection
<i>DiskOverheadInst</i>	CPU overhead for performing disk I/O
<i>FixedMsgInst</i>	Fixed number of instructions to process message
<i>MinDiskTime</i>	Minimum time required to access disk (in msec.)
<i>MaxDiskTime</i>	Maximum time required to access disk (in msec.)
<i>NetworkBandwidth</i>	Network Bandwidth (in Mbps.)
<i>PerByteMsgInst</i>	Number of instructions per message byte
<i>ServerBufSize</i>	Buffer size of server in pages
<i>Server MIPs</i>	Instruction execution rate of the server (in MIPs)
<i>SystemOverhead</i>	Number of instructions for certain system operations

Table 6.2. Physical Resource Parameters

network access requests of messages. The network is a shared resource and only one message can be transmitted at a time. Other messages are blocked until the current transmission is completed. The speed of the communication is determined by the network bandwidth. There are some additional costs of transmitting a message. One such cost is simulated using the parameter *FixedMsgInst* which denotes the fixed number of instructions to be executed in sending/receiving a message over the network. Besides this message processing overhead, variable number of instructions depending on the message size is modeled by the parameter *PerByteMsgInt*. The size of a control message, that is a non-data message like commit, abort, lock-request, etc., is specified by *ControlMsgSize*. The other possible overheads like lookup page copies, register locks, etc., are modeled by a single parameter, for both the clients and the server, which is *SystemOverhead*.

The physical resource model and overhead parameters are listed in Table 6.2.

6.5 Workload Models

The system workload is generated by client workstations. Client workstations continuously generate a stream of transactions after each exponential think time period with a mean of *ThinkTime*. The workload of the server is determined by the requests coming from the clients. The server responds to these requests in a FIFO order.

PARAMETER	MEANING
<i>ColdAccessProb</i>	Probability of access to a page in the cold region
<i>ColdBounds</i>	Page bounds in the cold region
<i>ColdWriteProb</i>	Probability of writing to a page in the cold region
<i>HotAccessProb</i>	Probability of access to a page in the hot region
<i>HotBounds</i>	Page bounds in the hot region
<i>HotWriteProb</i>	Probability of writing to a page in the hot region
<i>NoOfClients</i>	Number of clients
<i>ReadPageInst</i>	Number of instructions per page on read
<i>TransactionSize</i>	Number of pages accessed per transaction
<i>ThinkTime</i>	Mean think time between client transactions
<i>WritePageInst</i>	Number of instructions per page on write

Table 6.3. Workload Parameters

The number of pages accessed by a transaction is specified by *TransactionSize*. Each client spends an average CPU time of *ReadPageInst* for reading a page and *WritePageInst* for updating a page. According to the workload model used, there exists a *HotRegion* in the database for each client, where most of the references are made. The rest of the database is called the *ColdRegion*. With a probability of *HotAccessProb*, a page is chosen from the hot region of the database, and this page is updated with a probability of *HotWriteProb*. If the page is chosen from the *ColdRegion* with a probability of *ColdAccessProb*, it will be updated with a probability of *ColdWriteProb*. Table 6.3 presents the parameters to model the workload of the system.

There are different workloads used in the simulation model. Their parameter settings are provided in Table 6.4. The same parameter settings were used in the simulation experiments of [5]. The following subsections explain the workloads used in the simulation model.

PARAMETER	HOTCOLD	PRIVATE	UNIFORM	HICON
<i>TransactionSize</i>	20 pages	16 pages	20 pages	20 pages
<i>HotBounds</i>	p to p+49 $p=50(n-1)+1$	p to p+24 $p=25(n-1)+1$	-	1 to 250
<i>ColdBounds</i>	rest of DB	626 to 1250	all DB	rest of DB
<i>HotAccessProb</i>	0.8	0.8	-	0.8
<i>ColdAccessProb</i>	0.2	0.2	1.0	0.2
<i>HotWriteProb</i>	0.2	0.2	-	0.0 to 0.5
<i>ColdWriteProb</i>	0.2	0	0.2	0.2

Table 6.4. Workload Parameter Settings for Client n

HOTCOLD Workload

HOTCOLD workload is used to examine the performance of the algorithms when data accesses have high locality of reference¹ and there are moderate level of data conflicts. There is a high locality at each client in HOTCOLD workload, while there is a moderate read/write sharing among the clients. Each client has a 50 page *HotRegion*, where their 80% of the page accesses are directed. The remaining 20% of the accesses are directed to the rest of the database. The write probability to the accessed page is 0.2. It is the same for both of the database regions. *ColdRegion* of a client resides in the *HotRegions* of other clients.

PRIVATE Workload

PRIVATE workload is used to model systems like CAD, where clients work on their own set of data, while just reading the other parts of the database. In PRIVATE workload each client has 25 pages, where their 80% of the page accesses are directed. The remaining 20% of the accesses are directed to the rest of the database, where no other client's *HotRegion* resides. Clients can modify the pages in their *HotRegion* with a probability of 0.2, while they can only make read accesses to their *ColdRegion*.

¹If a data is accessed in the database, the next data access location will be close to the previous one (probably in the same page).

UNIFORM Workload

UNIFORM workload is used to examine the performance of the algorithms in terms of low locality and moderate conflict resolution. Clients have no *HotRegions*. They access all of the database with a uniform distribution. They update 20% of the pages that they accessed.

HICON Workload

HICON is a workload with varying degrees of data contention. It is often used to model shared-disk transaction processing systems. All of the clients have only one *HotRegion*, where their 80% of the page accesses are directed. The remaining 20% of the accesses are directed to the rest of the database. The write probability to the accessed page in *HotRegion* is uniformly distributed between 0.0 and 0.5, while the write probability to the accessed page in the *ColdRegion* is 0.2.

Chapter 7

Performance Experiments

This chapter presents the simulation results for the performance of the global memory management algorithms CB-A, FWD-HS, and FWD-SFD described in Chapter 5. The client-server simulation model described in Chapter 6 is used to test these algorithms. The performance of the algorithms is tested in four different workloads: HOTCOLD, PRIVATE, UNIFORM and HICON. The system parameters and the workloads are described in Section 6.5. Table 7.1 lists the simulation parameter settings used in these experiments and Table 6.4 lists the workload parameter settings used in these experiments.

Following is the list of performance metrics used for the analysis of experimental results.

- **Throughput:** is the number of the committed transactions per second.
- **Portion of the database available in the global memory:** refers to the portion of the database that each client in the system can access without any disk I/O. It is calculated as the fraction of union size of the database pages in the clients' and the server's buffers with respect to the whole database size.
- **Total disk I/O per commit:** is the total number of the disk I/Os performed in the system for each committed transaction.
- **Messages sent per commit:** is the total number of the control and

PARAMETER	SETTING
<i>ClientBufSize</i>	5%, 10%, 25%, or 50% of DB
<i>ClientMip</i>	50 MIPs
<i>ControlMsgSize</i>	256 bytes
<i>DatabaseSize</i>	1250
<i>DeadlockInterval</i>	1 second
<i>DiskOverheadInst</i>	5,000 instructions
<i>FixedMsgInst</i>	20,000 instructions
<i>MaxDiskTime</i>	30 milliseconds
<i>MinDiskTime</i>	10 milliseconds
<i>NetworkBandwidth</i>	8 or 80 Mbits/sec
<i>NoOfClients</i>	1 to 25
<i>PageSize</i>	4,096 bytes
<i>PerByteMsgInst</i>	10,000 inst. per 4Kb
<i>ReadPageInst</i>	30,000 instructions
<i>ServerBufSize</i>	10%, 50% or 100% of DB
<i>Server MIPs</i>	100 MIPs
<i>SystemOverhead</i>	300 instructions
<i>ThinkTime</i>	0 milliseconds
<i>WritePageInst</i>	60,000 instructions

Table 7.1. Simulation Parameter Settings

data messages sent for each committed transaction.

- **Message volume per commit:** is the total number of the message bytes sent through the network for each committed transaction.
- **Server buffer hit ratio:** is the ratio of the server buffer hits over the total number of pages accessed by the server.
- **Server misses forwarded:** is the ratio of the page request messages forwarded to other clients over the total number of server misses.
- **Dropped pages per commit:** is the total number of pages dropped for each committed transaction.
- **Pages kept in memory per commit:** is the total number of the dropped pages that should be kept in the global memory (either in server's buffer or in another client's buffer), since they are the only copies.

The following sections analyze the performance of the algorithms in different workloads, with respect to the performance metrics described above.

7.1 HOTCOLD Workload

The first set of performance results are obtained with the HOTCOLD Workload. Recall that in the HOTCOLD workload (see Section 6.5), each client has a 50 page HotRegion where 80% of the accesses are directed, and the rest of the database is the ColdRegion where the remaining 20% of the accesses are directed. The HotRegion of a client is in the ColdRegion of other clients. The transactions update 20% of the accessed pages in both regions. The following subsections explain the results of the experiments in terms of the portion of the database available in global memory, the resource requirements, and the throughput of the system. For the analysis of the portion of the database in memory and the resource requirements, only one set of experimental results are used (5% client buffer size, 50% server buffer size, and slow network speed). Other results are not shown in this chapter, as they follow the same characteristics as the one used, except that they are scaled according to the buffer sizes and the network speed. Therefore, the discussions of the results in the given set of experiments can also be applied to the other set of experiments.

7.1.1 Portion of Database Available in Memory

Figure 7.1 shows the percentage of the database available in the global memory for the algorithms running under the HOTCOLD workload. The highest portion of the database that can be available in the global memory is shown with dotted lines. The database portion in the algorithms deviate from the ideal condition because of the replicated data that reside in the global memory. There can be two forms of data-replication in the global memory[5]:

- **Server-client correlation:** the data has a local copy both in the server, and the client. Clients request pages from the server. The server reads these pages from disk into its buffer and sends them to the clients. Therefore the data replications between the clients and the server occurs. The

number of the replicated pages in small systems is greater than the number in large systems. As more clients are added to the system, the ratio of the pages at the server to the pages at the clients becomes smaller, and thus the number of the replicated pages in the server's buffer and the clients' buffers decreases. Sending the updated pages to the server at the end of the commit time also increases the server-client correlation.

- **Client-client replication:** the data has local copies in the different clients. When the transactions running on different clients access the same database page, the page is replicated in these clients. As the number of clients increases the page conflicts increase and more callback messages are sent to the clients reducing the number of the client-client replication.

As it can be seen in Figure 7.1, CB-A has the smallest portion of the database in the global memory, because of the server-client correlation. The server sends the page to the client, and the copy of that page is kept in server's buffer with the MRU value. As the number of the clients increases, the replicated pages that reside both in the client and the server decrease, and the portion of database in the global memory increases. Client-client replication does not effect CB-A, since each client accesses only its own buffer and the server's buffer. The algorithms CB-R and FWD-HS (with 30% server buffer size) are compared in [5]. The performance of CB-R seems to be independent of the number of the clients added to the system. However the database ratio increases with CB-A. This is because of the implementation differences in CB-R and CB-A algorithms (see Section 3.2.1). Recall that at the end of each transaction, the write intentions are dropped in CB-R, but they are kept in CB-A until they are called back or the client decides to drop it. In CB-R when a client receives a callback it drops the page out of its buffer, while the write intention is downgraded to read intention in CB-A. The page is not dropped in CB-A. Therefore, the portion of the database in CB-A is greater than the portion in CB-R. However CB-A has still the least value when it is compared with FWD-HS and FWD-SFD.

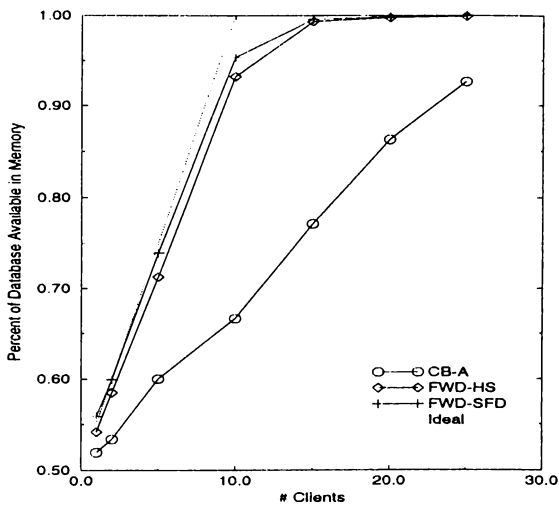


Figure 7.1. % of DB Available in Memory (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

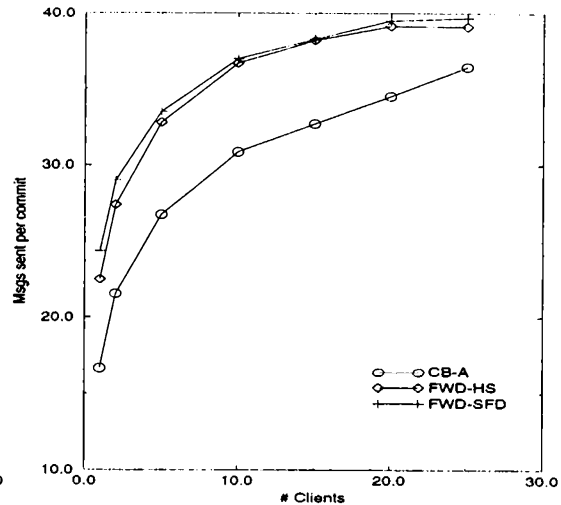


Figure 7.3. Messages Sent per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

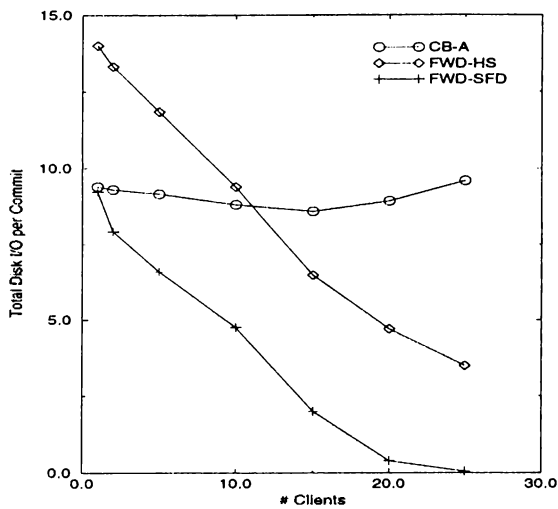


Figure 7.2. Total Disk I/O per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

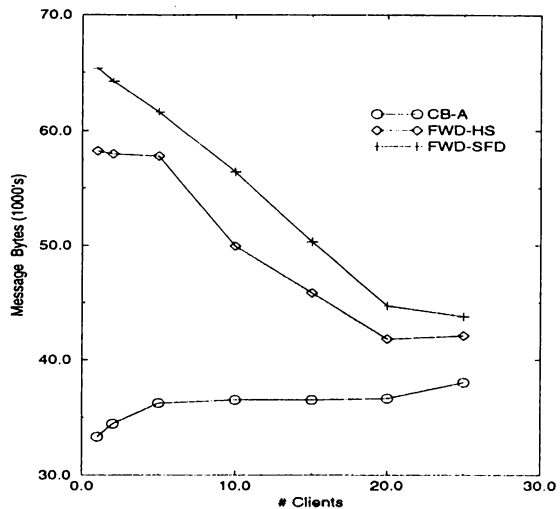


Figure 7.4. Message Volume per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

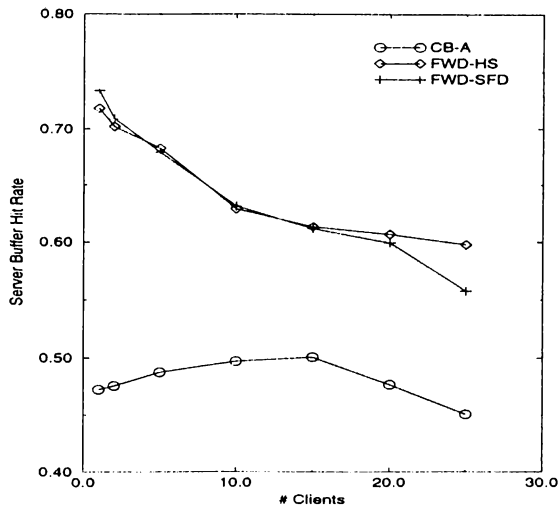


Figure 7.5. Server Buffer Hit Ratio (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

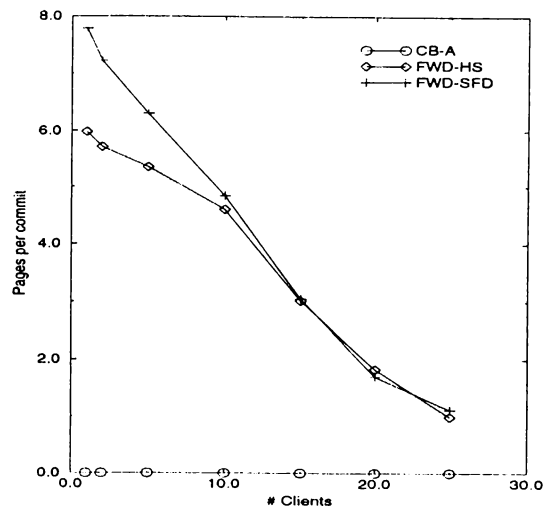


Figure 7.7. Dropped Pages Kept in Memory per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

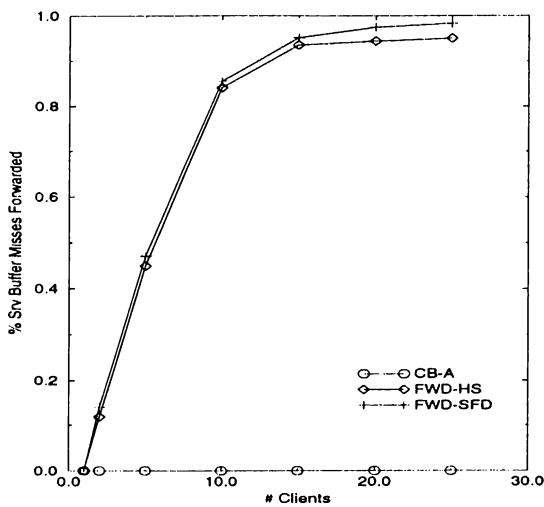


Figure 7.6. Server Misses Forwarded (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

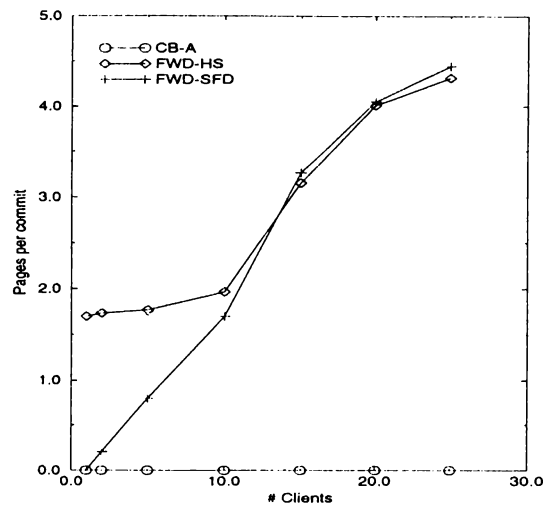


Figure 7.8. Dropped Pages per Commit (HOTCOLD, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

FWD-HS and FWD-SFD are very close to the ideal case and they approach the 100% ratio when the number of clients increases. This is caused by the client-client replication and the server-client correlation. Because of the HOT-COLD workload, clients share pages in the database. Overlapping transactions in different clients may request the same page. In FWD-HS, when a page is sent to a client, the server marks the page as LRU. If a need for an empty buffer location occurs in the server, the server just drops the sent page out of its buffer. Then the replicated copy is removed out of the global memory. Therefore the shared copies of the same page both in the client and the server decreases, and the portion of the database in global memory increases. Sending Dropped Pages technique also increases the portion of the database in global memory, as when the only local copy of a page is dropped, it is sent to the server.

FWD-SFD shows a little improvement in the portion of the database in global memory, when compared with FWD-HS. There are two reasons why FWD-SFD provides better results than FWD-HS. The first reason is that, when the server sends a page to a client, it just drops the page out of its buffer, reducing the number of copies that reside both in the client and the server. The other reason is that, in FWD-HS, when a client is going to drop the only local copy of a page, the page is sent to the server. However in FWD-SFD, buffers of other clients can be exploited; i.e., the dropped pages can also be sent to other clients. FWD-SFD utilizes the global memory better than the other two algorithms, when the number of clients is small. As stated at the beginning of the section when the number of clients increases, the server-client correlation decreases and the results obtained for the portion of the database available in FWD-HS and FWD-SFD become similar.

7.1.2 Resource Requirements

Recall that the major aim of the global memory management algorithms is to reduce disk I/O while increasing the portion of the database in the global memory. As the portion of the database increases, the probability that an accessed page will be found in the global memory increases, and the total number of disk I/Os decreases. Figure 7.2 shows the total number of disk I/Os needed for a transaction to complete. The message requirements are

shown in Figures 7.3 and 7.4. The message and disk I/O requirements for each committed transaction heavily depend on the client buffer hit ratio, the server buffer hit ratio, the percentage of the server misses forwarded to other clients, total number of dropped pages, and finally the total number of dropped pages kept in memory.

The client cache hit ratio is similar in all clients which is about 69%, since all the algorithms are based on CB-A and the global memory management techniques do not affect buffering at clients. The client hit ratio of FWD-HS as obtained in [5]¹ is also constant (65%). As the hit ratio in client caches increases, the disk I/O decreases. Server buffer hit ratio (shown in Figure 7.5) is another parameter that affects the disk I/O. If the server hit ratio increases, then the disk I/O decreases. CB-A algorithm has the smallest server hit ratio, as the portion of the database in the global memory is low. In CB-A, the pages that are sent to the clients are not marked as LRU. They are not dropped out of the global memory for the first empty space requirements in the server's buffer. Also the page requests are not forwarded to other clients. Server hit ratio follows the same pattern in both FWD-HS and FWD-SFD when the number of clients is less than 15 (i.e., before the portion of the database available in the global memory reaches 100%). As the number of the clients increases beyond that point, server hit ratio of FWD-SFD decreases faster. Some of the dropped pages are forwarded to the clients that have free space in their buffers. Therefore, the server does not keep all the dropped pages in its buffer. It can be seen in Figure 7.6 that the number of page requests forwarded in FWD-SFD is greater than that in FWD-HS. The difference increases when the number of clients is greater than 15.

In all of the algorithms, clients send the same number of page requests to the server. CB-A has the least number of messages sent for each committed transaction, as it does not employ the Forwarding and Sending Dropped Pages techniques (see Chapter 5). FWD-SFD has slightly more messages per commit than FWD-HS. Forwarding Dropped Pages technique increases the number of messages exchanged. There are two different types of messages exchanged in the system. One is the control message that is small in size (e.g., PageRequest, LockRequest messages, etc.), and the other one is the data message that is

¹Recall that the server buffer size is 30% and the CB-R algorithm is used as the base algorithm.

large in size (i.e., page message). The message volume is the message bytes sent per committed transaction. If more page messages are sent, the message volume becomes larger. When the number of clients is small, the portion of the database available is small, and the probability that the dropped page is the only local copy is high (see Figure 7.7). Therefore the page is not dropped, and it is kept in the global memory with FWD-HS and FWD-SFD. As more dropped pages are kept in the global memory, more database pages are exchanged and the message volume is increased. When the number of clients is large, there are more than one local copy of the dropped page either in any of the clients or in the server. Therefore, most of the pages are dropped (see Figure 7.8) and the message volume is decreased. When the server hit ratio decreases, the total number of messages sent per commit increases with FWD-HS and FWD-SFD, since a portion of the page requests are forwarded to other clients.

If we compare the results obtained for the total number of messages exchanged in FWD-HS with those obtained in [5], we see that the number of messages transmitted is smaller in [5] when the number of clients is small. The server buffer that we have used in our experiments (50% and 30%) is greater than the one used in [5]. Therefore, less number of page requests is forwarded as the server buffer hit ratio increases. Besides, CB-A keeps a larger portion of the database in the global memory. On the other hand, CB-A requires more messages to be sent, as the obtained write intentions are not released at the end of a transaction. As a result, more conflicting lock requests occur, and more callback messages are sent to the clients.

Total disk I/O (see Figure 7.2) results are similar to those obtained in [5]. At the end of each transaction, dirty pages are sent to the server increasing the total disk I/O. As the number of clients increases, the number of dirtied pages increases. The disk I/O has crucial effects on the performance of the algorithms. If the page requested from the server is not in the global memory the server reads the page from its disk. As the portion of the database in the global memory increases, the disk I/Os due to page requests are decreased. Recall that FWD-SFD has smaller number of disk I/Os than FWD-HS. FWD-SFD does not have less disk writes, but the difference occurs due to disk reads. Sending dropped pages to the server also increases the disk I/O. If there is no space in the server's buffer, the server has to replace a page with the newly received one. These kinds of disk writes are also decreased in FWD-SFD, as

the dropped pages are forwarded to empty clients, when the server buffer is full.

7.1.3 Throughput Results

Different system parameters that affect the throughput are experimented. These parameters are:

- Client buffer size,
- Server buffer size,
- Network speed.

The results that are obtained under different conditions regarding those parameters are discussed in the following subsections.

The effects of client buffer size under slow network:

Four different client cache sizes are experimented to investigate the effect of the client cache size on the throughput of the algorithms. The performance results are obtained under both the slow network and the fast network.

Buffer sizes range as 5% of the database size (shown in Figure 7.9), 10% of the database size (shown in Figure 7.10), 25% of the database size (shown in Figure 7.11), and 50% of the database size (shown in Figure 7.12) where the server buffer size is fixed to 50% of the database size, and the network is slow (i.e. , *NetworkBandwidth* is 8 Mbits/sec). FWD-HS and FWD-SFD outperform CB-A, for all experimented client buffer sizes. This is because of the high disk I/O. There is a small performance improvement in FWD-SFD over FWD-HS, when the client buffer sizes is 5% and 10% of the database, as the portion of the database available in the global memory is high with FWD-SFD. If the client buffer size is 25%, FWD-HS and FWD-SFD give nearly the same throughput results. The extreme case where the client buffer sizes are equal to the server buffer size is also tested. In this experiment FWD-HS

and FWD-SFD exhibit the same performance. As the buffer size increases, larger portion of the hot data is put in the buffer, and the performance of the algorithms becomes better. The throughput results show the same pattern in FWD-HS, when compared with the results in [5].

The effects of client buffer size when the network is fast:

Figures 7.13 to 7.16 show the throughput results when the network is fast (i.e., *NetworkBandwidth* is 80 Mbits/sec). Under speed network, FWD-HS and FWD-SFD algorithms again outperform the CB-A algorithm. FWD-SFD provides better performance than FWD-HS. When the network is fast, total disk I/O more strongly affects the results in throughput. Having less disk I/O, FWD-SFD performs better. When the client cache size increases, the difference between the performances of FWD-HS and FWD-SFD decreases.

The effects of server buffer size when the network is slow:

The effects of server buffer size are experimented under the two extreme cases where the server buffer size is 10% and 100% of the database size. The results obtained using the slow network are shown in Figures 7.17 and 7.18, and they are compared with the results that are obtained when the server buffer size is 50% (see Figure 7.10). In these experiments, the client buffer size is fixed to 10%.

FWD-SFD performs the best when the server buffer size is small. FWD-SFD exploits the clients buffer for sending the dropped pages. If a server has small buffer, in FWD-HS all the dropped pages received by the server causes page replacement in the server's buffer pool. Therefore, the performance of FWD-HS becomes worst with increased disk I/O.

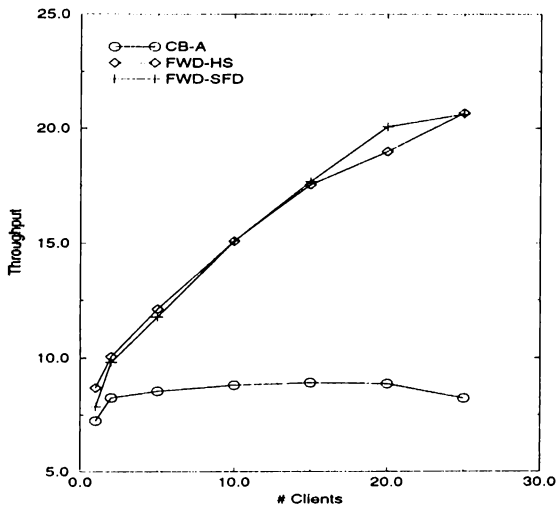


Figure 7.9. Throughput (HOTCOLD, 50% Ser Buf, 5% Cli Bufs, Slow Net).

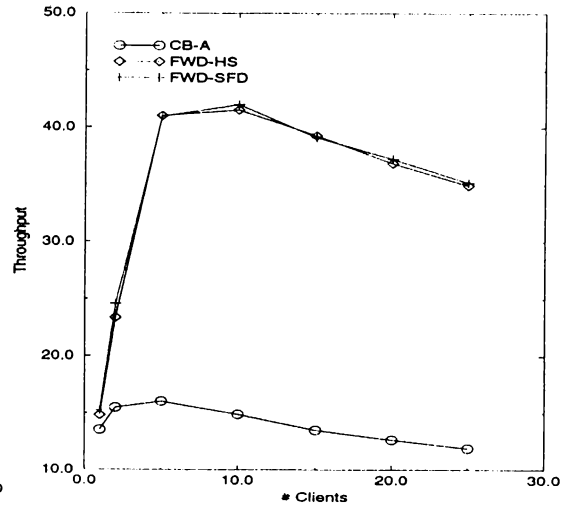


Figure 7.11. Throughput (HOTCOLD, 50% Ser Buf, 25% Cli Bufs, Slow Net).

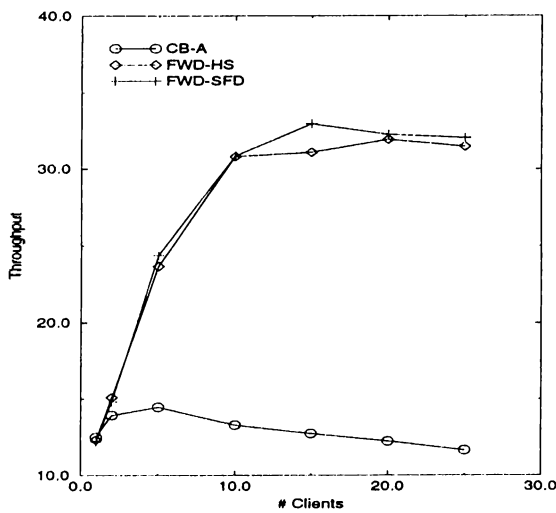


Figure 7.10. Throughput, (HOTCOLD, 50% Ser Buf, 10% Cli Bufs, Slow Net).

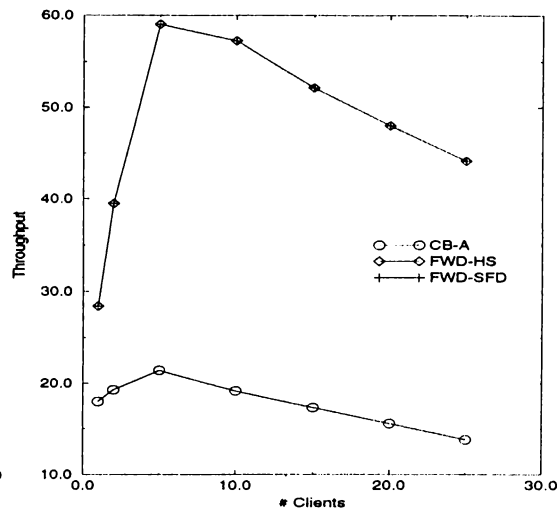


Figure 7.12. Throughput (HOTCOLD, 50% Ser Buf, 50% Cli Bufs, Slow Net).

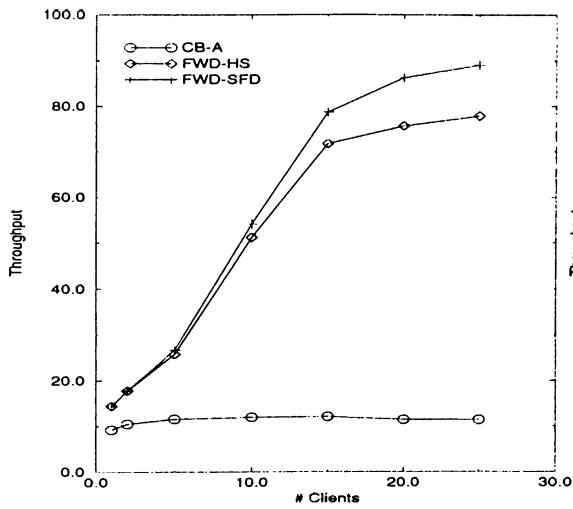


Figure 7.13. Throughput (HOT-COLD, 50% Ser Buf, 5% Cli Bufs, Fast Net).

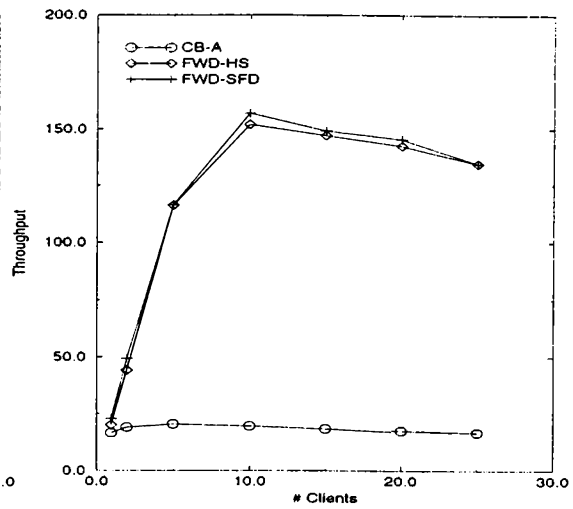


Figure 7.15. Throughput (HOT-COLD, 50% Ser Buf, 25% Cli Bufs, Fast Net).

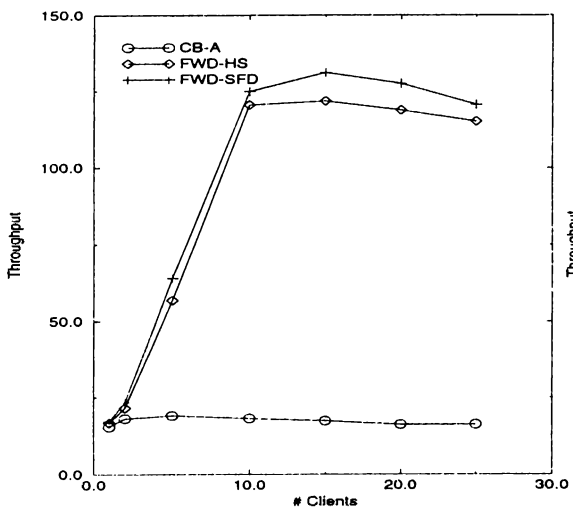


Figure 7.14. Throughput (HOT-COLD, 50% Ser Buf, 10% Cli Bufs, Fast Net).

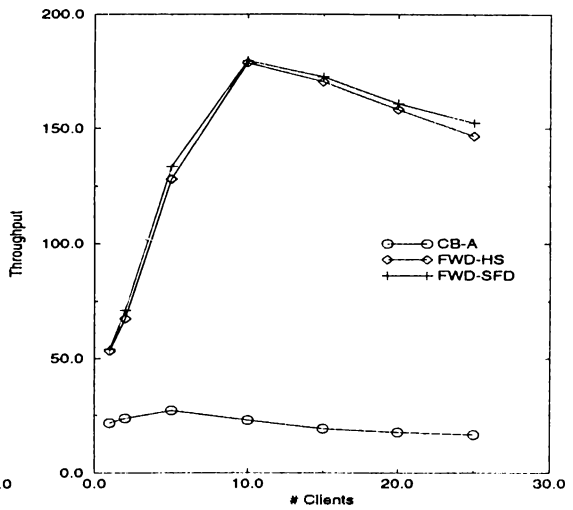


Figure 7.16. Throughput (HOT-COLD, 50% Ser Buf, 50% Cli Bufs, Fast Net).

When the server buffer is large (i.e., 100%), CB-A performs better than the other two algorithms. The server has all the database in its buffer, and none of the algorithms used to improve the portion of the database in the global memory is needed. The performance of FWD-SFD is very low compared to the other two when the number of clients is small. The server drops the sent pages out of its buffer in order to open space for the other dropped pages. However, there is no need for this operation, as the server buffer is large enough to hold the whole database. Applying global memory management techniques is redundant. Even these techniques decrease the performance of the system. When the number of clients increases, dropped pages out of server buffer do not effect the performance of FWD-SFD. Dropping the pages also increases the probability that a page that is going to be dropped by a client will be the only copy in the global memory. When the number of clients is small, this probability is higher. Therefore, redundant operations are done when the server buffer size is large. When the number of clients increases, this probability decreases, and the performance of FWD-SFD becomes better, but it is still lower than that of the other two algorithms.

The effects of server buffer size when the network is fast:

With the fast network, the results shown in Figures 7.14, 7.19, and 7.20 are obtained. The results show similar characteristics with the results obtained with the slow network when the server buffer size is 10% or 50% of the database size. Because of the fast network, the throughputs are higher with all three algorithms. However, when the server buffer size is 100%, the improvement in the performance of FWD-SFD is at a higher level compared to the other algorithms. The effect of redundant messages sent over the network is reduced, because of the high speed of the network. All three algorithms show similar performance characteristics, however, the performance of CB-A is a little bit better than the performance of other two algorithms.

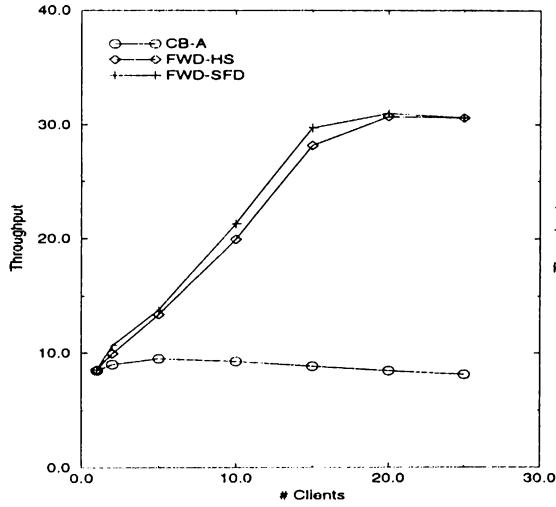


Figure 7.17. Throughput (HOT-COLD, 10% Ser Buf, 10% Cli Bufs, Slow Net).

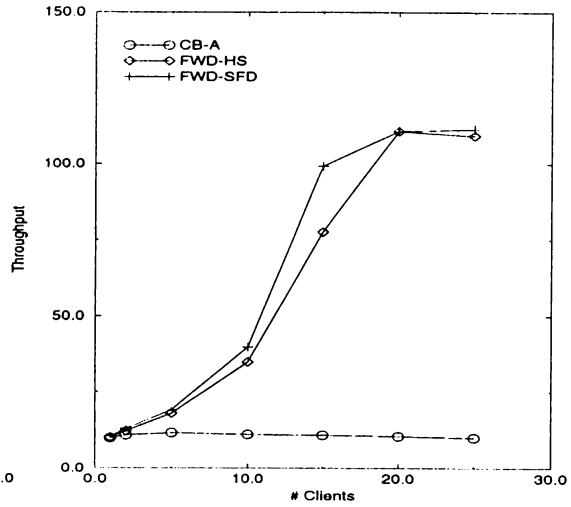


Figure 7.19. Throughput (HOT-COLD, 10% Ser Buf, 10% Cli Bufs, Fast Net).

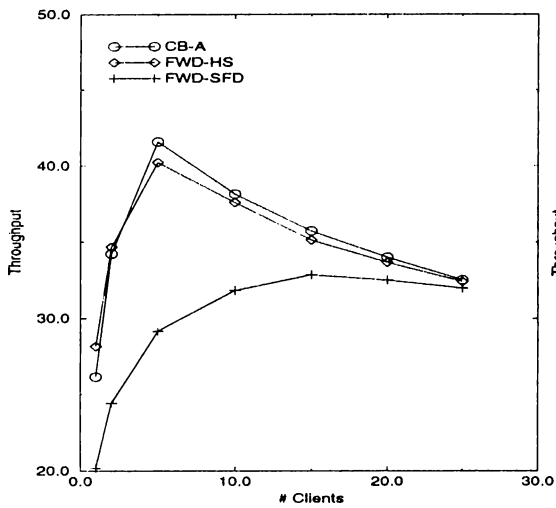


Figure 7.18. Throughput (HOT-COLD, 100% Ser Buf, 10% Cli Bufs, Slow Net).

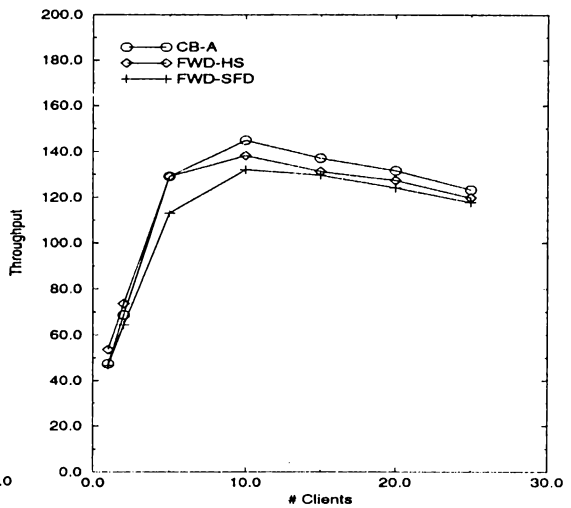


Figure 7.20. Throughput (HOT-COLD, 100% Ser Buf, 10% Cli Bufs, Fast Net).

7.2 PRIVATE Workload

This section discusses the performance results obtained under the PRIVATE workload. Recall that in the PRIVATE workload (see Section 6.5), each client has a 25 page HotRegion where 80% of the accesses are directed, and the remaining 20% of the accesses are directed to the ColdRegion. The transactions update 20% of the accessed pages that reside only in their HotRegions. Therefore, there is no read/write sharing of data and no data contention in this workload. The experiments differ from [5] in the following:

- The basic algorithms is CB-A (not CB-R).
- Server buffer size is 50% (not 30%).
- *HotAccessProb* and *ColdAccessProb* are respectively 0.8 and 0.2 (not 0.5 for both regions).

As a result of these differences, the results of our performance experiments are not similar to those obtained in [5]. Figures 7.21 to 7.24 display the performance results of the algorithms when the network is slow, and the server buffer size is the 50% of the database size. In all of the figures, when the number of clients is less than 10, the throughput increases as the number of clients increases (similar to the results obtained with the HOTCOLD workload). When the client buffer size is small (i.e., 5% of the database size), CB-A exhibits better performance than FWD-HS and FWD-SFD, since most of the pages in the HotRegion are kept in clients' buffers and the pages in the ColdRegion are kept in the server's buffer (50% of the database is the ColdRegion for all the clients). As it can be seen in Figure 7.25, the server buffer hit ratio of CB-A is almost 1.

The server buffer hit ratio of FWD-HS and FWD-SFD are the same when the number of clients is one, and, as the number of clients increases the hit ratios of these algorithms decrease, because of the Hate Hints and the Dropping Sent Pages techniques. Updated HotRegion pages are sent to the server at the end of each transaction. These pages are marked as the MRU in the server. Therefore, the ratio of the hot pages in the server increases. With CB-A, the pages sent to the clients are marked as MRU which causes the hot pages to

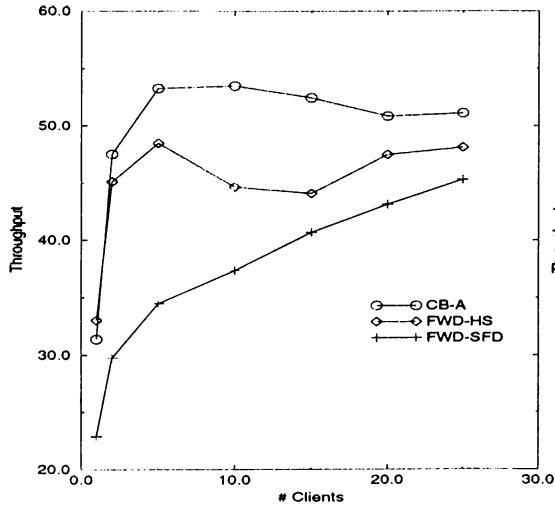


Figure 7.21. Throughput (PRIVATE, 50% Ser Buf, 5% Cli Bufs, Slow Net).

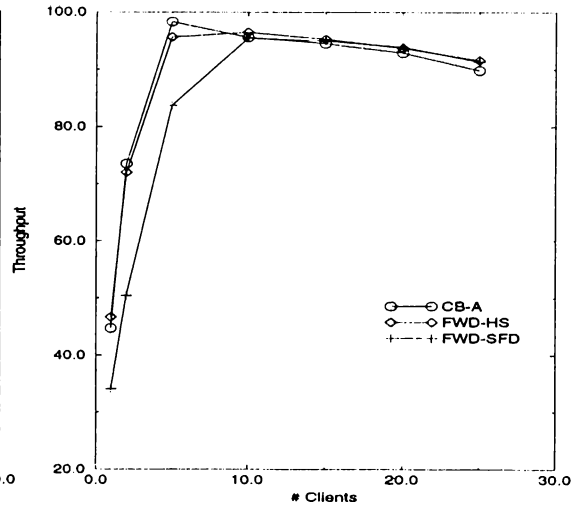


Figure 7.23. Throughput (PRIVATE, 50% Ser Buf, 25% Cli Bufs, Slow Net).

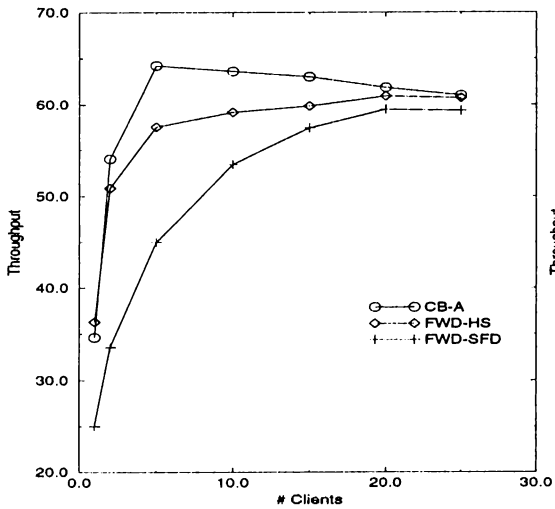


Figure 7.22. Throughput (PRIVATE, 50% Ser Buf, 10% Cli Bufs, Slow Net).

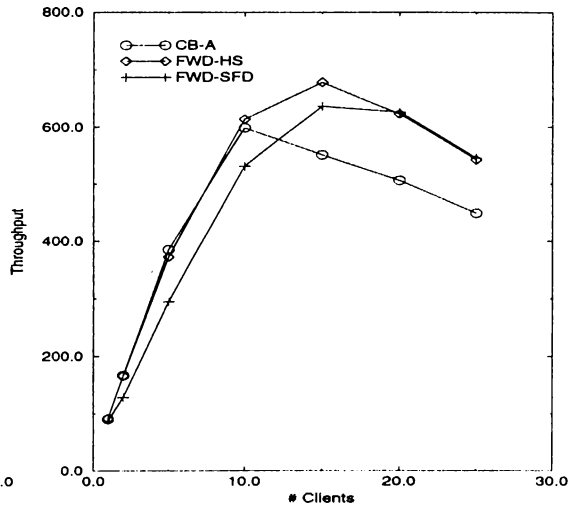


Figure 7.24. Throughput (PRIVATE, 50% Ser Buf, 50% Cli Bufs, Slow Net).

become the LRU and to be dropped out of the server buffer. However, with FWD-HS and FWD-SFD, the hot pages are not dropped out of the server buffer. In FWD-HS, the pages sent are marked as the LRU, and in FWD-SFD, these pages are just dropped. Figure 7.21 shows that FWD-HS exhibits better performance than CB-A when there is only one client in the system, as it does not allow the only copy of the pages to be dropped out of the global memory. However, when there is more than one client in the system, the CB-A provides better performance than FWD-HS, since the server hit ratio of CB-A is higher than FWD-HS.

Continuing with Figure 7.21, it can be seen that FWD-HS provides an increase in the performance as the number of clients increases until it becomes 5. This is due to the increase in the concurrency of the executing transactions. However, when the number of clients increases beyond 5, there is a decrease in the performance since the total number of dropped pages kept in the global memory increases (see Figure 7.26). As the number of clients increases, more updated pages are sent to the server and more cold pages are dropped out of the server's buffer, so that the probability that the dropped page is the only local copy in the global memory is increased. This causes more dropped pages to be sent to the server and the message volume (see Figure 7.27) is increased. This is very important when the network is slow. The throughput of FWD-HS is decreased because of the Sending Dropped Pages technique. As the number of clients increases, the portion of the database in the global memory increases, and fewer number of dropped pages are sent to the server. As it can be seen in Figure 7.28, more pages are dropped. The message volume is decreased and the throughput is increased.

FWD-SFD exhibits the worst performance, since it drops the sent pages out of the server's buffer. This is redundant, because there is no need to increase the empty space in the server's buffer since the server's buffer size is large enough to keep the all ColdRegion pages.

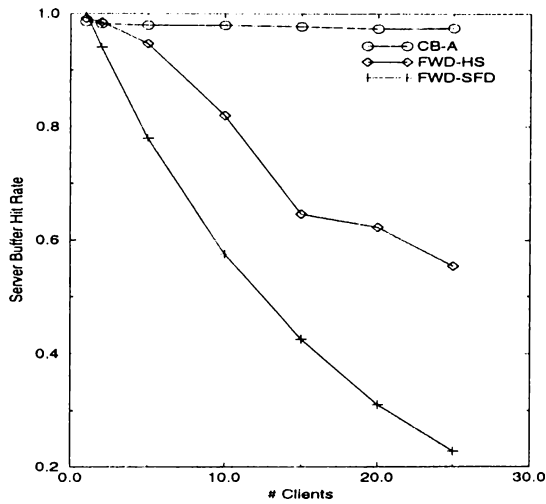


Figure 7.25. Server Buffer Hit Ratio (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

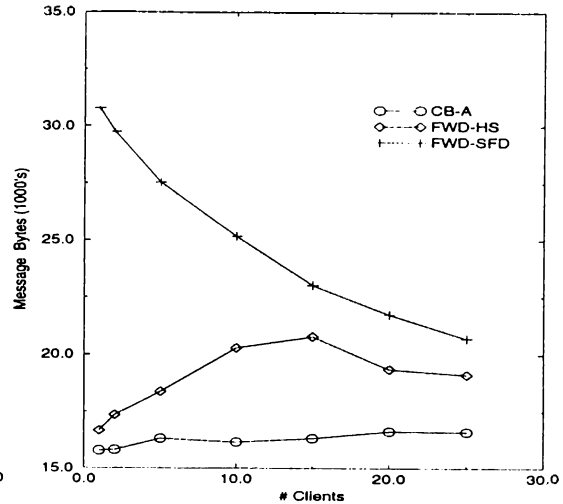


Figure 7.27. Message Volume per Commit (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

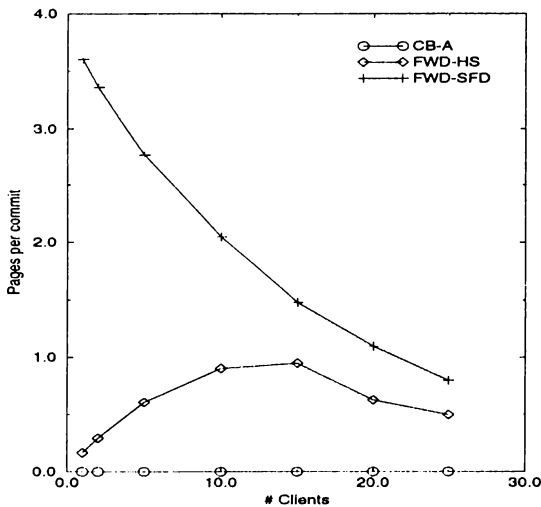


Figure 7.26. Dropped Pages Kept in Memory per Commit (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

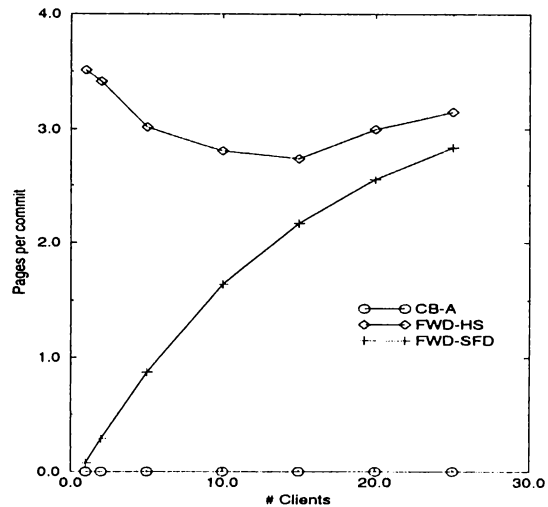


Figure 7.28. Dropped Pages per Commit (PRIVATE, 50% Ser Bufs, 5% Cli Bufs, Slow Net).

When the client buffer size increases, the performances of the three algorithms become similar, since the portion of the database in the global memory increases and less number of dropped pages are kept in the global memory. In the extreme case where the client buffer size is 50% and the number of clients is large, CB-A exhibits the same performance as FWD-HS until the number of clients becomes 10. After that point, the performance of the CB-A degrades, and it becomes the worst, since the server and the network become bottleneck. In the extreme case, FWD-SFD performs worse than FWD-HS, as the forwarding dropped pages does not improve the performance. The portion of the database available in the global memory is high enough such that no more dropped pages are kept in the global memory and no more disk I/Os required. Therefore with FWD-SFD, the Dropping Sent Pages out of the server's buffer, does no longer increase the performance, instead throughput of FWD-SFD decreases as the server hit ratio decreases and more page requests are forwarded. However, when the number of clients increases beyond 15 clients FWD-HS and FWD-SFD exhibit the same performance. After that value, their throughput values start decreasing. The server and the network become bottleneck, and more data conflicts occur.

Under the fast network (see Figures 7.29 to 7.32), the message volume becomes less important, and the total disk I/Os determine the throughput. When the number of clients is small, FWD-HS and CB-A exhibit better performance than FWD-SFD, since the portion of the database available in the global memory is small. However, when the number of clients increases, FWD-SFD provides better performance than the other two since higher portion of the database becomes available in the global memory, and the disk I/O is decreased. As expected, CB-A leads to the least throughput value under these conditions.

Server buffer size is another important parameter for the performance results. When the server buffer size is small, FWD-HS and FWD-SFD outperform CB-A, as they exploit the clients' buffers for page requests. When the fast network is used, FWD-SFD exhibits a little bit better performance than FWD-HS, because although the message volume associated with FWD-SFD is higher than that of FWD-HS, the disk I/O required for FWD-SFD is less than that of FWD-HS. When the server buffer size is increased to the ideal case (i.e., the buffer size becomes 100% of the database size), FWD-SFD has the

worst performance under the slow network since it drops the sent messages out of the server's buffer. Again, this is redundant as the server's buffer is large enough to hold all the database pages. Although CB-A provides better performance than FWD-HS, both algorithms behave similarly. The throughput with FWD-HS decreases because of the implementation of Hate Hints and Sending Dropped Pages. When the network becomes faster and the number of clients increases, the portion of the database available in the global memory increases and the time spent for the transmission of network messages decreases, so that FWD-SFD produces higher throughput values than the other two algorithms.

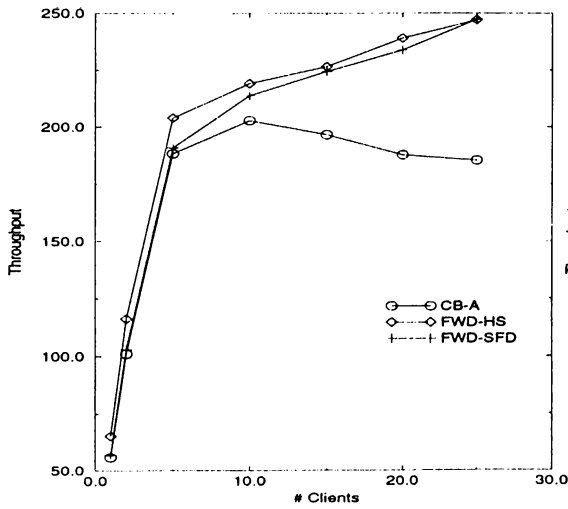


Figure 7.29. Throughput (PRIVATE, 50% Ser Buf, 5% Cli Bufs, Fast Net).

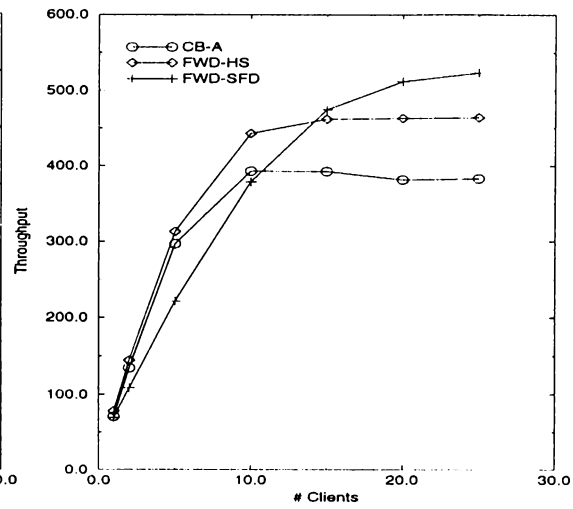


Figure 7.31. Throughput (PRIVATE, 50% Ser Buf, 25% Cli Bufs, Fast Net).

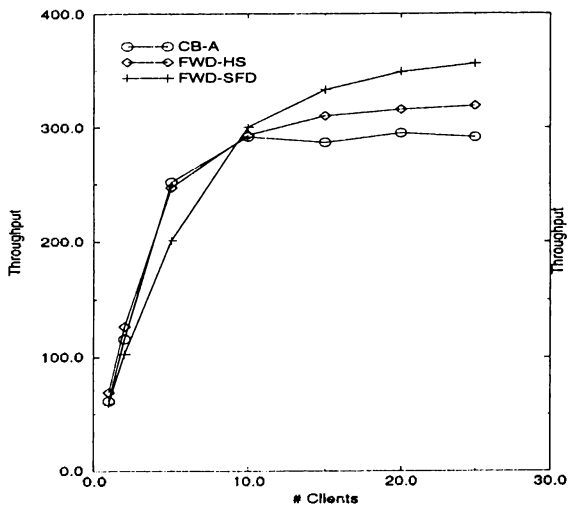


Figure 7.30. Throughput (PRIVATE, 50% Ser Buf, 10% Cli Bufs, Fast Net).

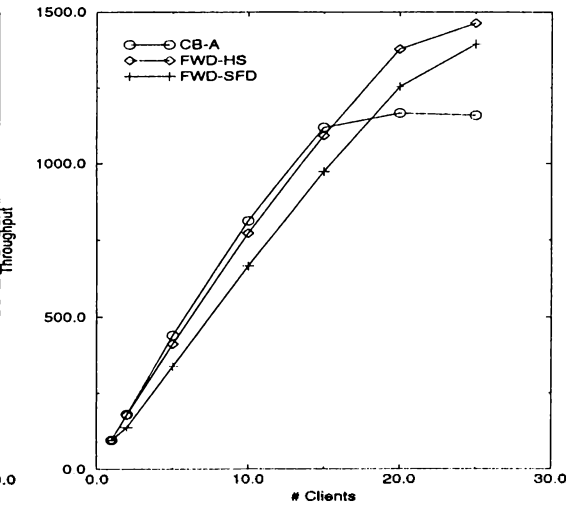


Figure 7.32. Throughput (PRIVATE, 50% Ser Buf, 50% Cli Bufs, Fast Net).

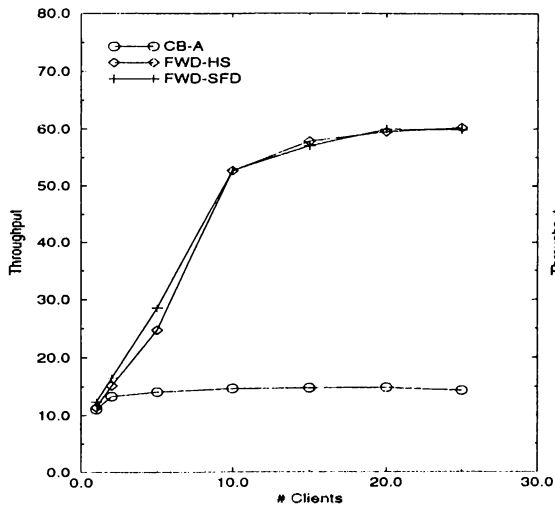


Figure 7.33. Throughput (PRIVATE, 10% Ser Buf, 10% Cli Bufs, Slow Net).

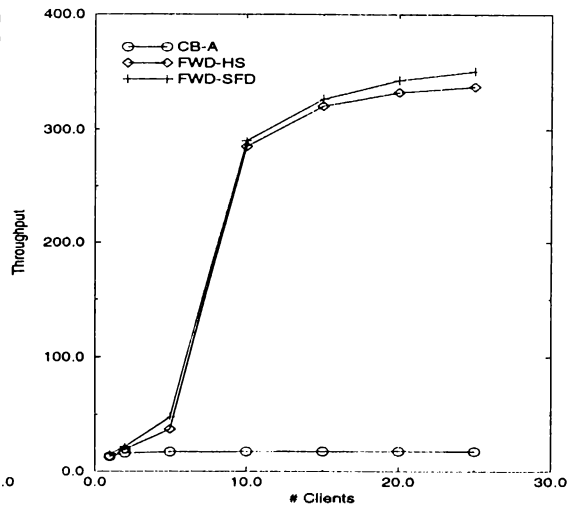


Figure 7.35. Throughput (PRIVATE, 10% Ser Buf, 10% Cli Bufs, Fast Net).

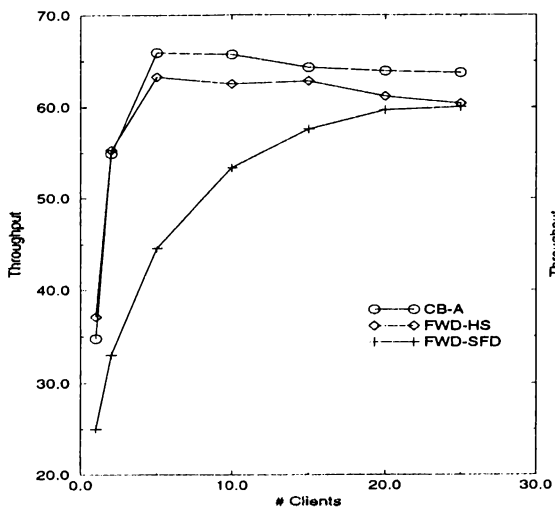


Figure 7.34. Throughput (PRIVATE, 100% Ser Buf, 10% Cli Bufs, Slow Net).

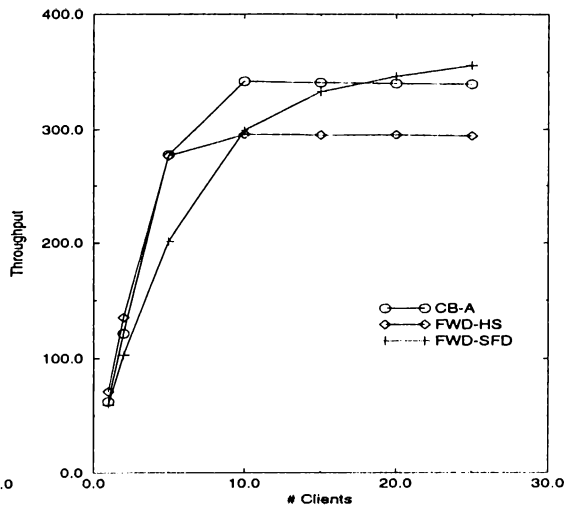


Figure 7.36. Throughput (PRIVATE, 100% Ser Buf, 10% Cli Bufs, Fast Net).

7.3 UNIFORM Workload

UNIFORM workload is another workload used in our experiments. As stated in Section 6.5, in UNIFORM workload all the database resides in the Cold-Region and is accessed uniformly. Figures 7.37 to 7.40 show the throughput results obtained under the slow network. As it can be seen from the figures, FWD-SFD and FWD-HS produce the same performance results and they both outperform CB-A. There is no HotRegion in this workload. Therefore, the lack of locality reduces the importance of the Sending Dropped Pages and the Forwarding Dropped Pages techniques, while the Hate Hints and the Dropping Sent Pages are still important, since they increase the portion of the database available in the global memory. The difference between the Hate Hints and the Dropping Sent Pages techniques becomes less important, as there does not exist any locality and it is more probable that with FWD-HS the sent page will be replaced in the next page request. However, as the number of clients increases, the Forwarding and the Forwarding Dropped Pages techniques increase the throughput results.

When a fast network is employed, with small client buffer sizes (see Figures 7.41 to 7.44) FWD-SFD exhibits better performance than FWD-HS, as observed with all other types of workload. However, when the client buffer size increases, the difference becomes less, since larger portion of the database becomes available in the global memory.

Server buffer size also affects the performance of the algorithms. FWD-SFD provides better performance than FWD-HS when the server buffer is small (see Figures 7.45 and 7.47). Increasing the network speed increases the performance of FWD-SFD a little bit, since more messages need to be exchanged with this algorithm. On the other hand, when the buffer size becomes equal to the 100% of the database size, FWD-SFD exhibits the worst performance under the slow network, since the server drops all the sent pages out of its buffer. CB-A exhibits the best performance as all database pages are kept in the server's buffer. However, under the fast network (see Figure 7.48), the performance of FWD-SFD becomes better as the number of clients increases. CB-A still provides better performance than FWD-HS.

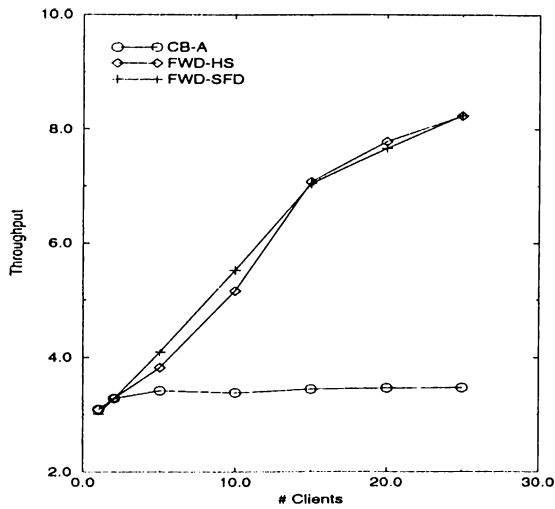


Figure 7.37. Throughput (UNIFORM, 50% Ser Buf, 5% Cli Bufs, Slow Net).

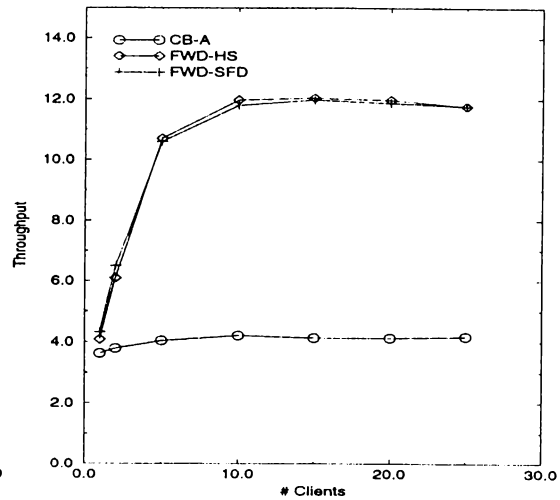


Figure 7.39. Throughput (UNIFORM, 50% Ser Buf, 25% Cli Bufs, Slow Net).

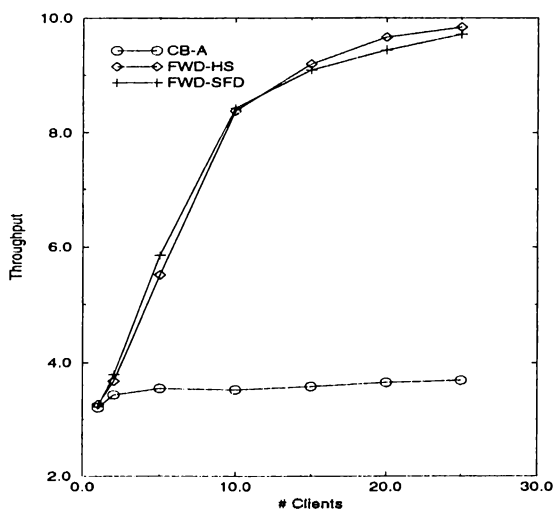


Figure 7.38. Throughput (UNIFORM, 50% Ser Buf, 10% Cli Bufs, Slow Net).

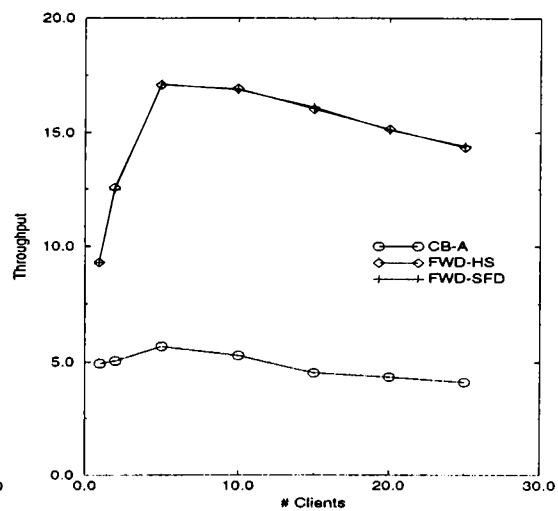


Figure 7.40. Throughput (UNIFORM, 50% Ser Buf, 50% Cli Bufs, Slow Net).

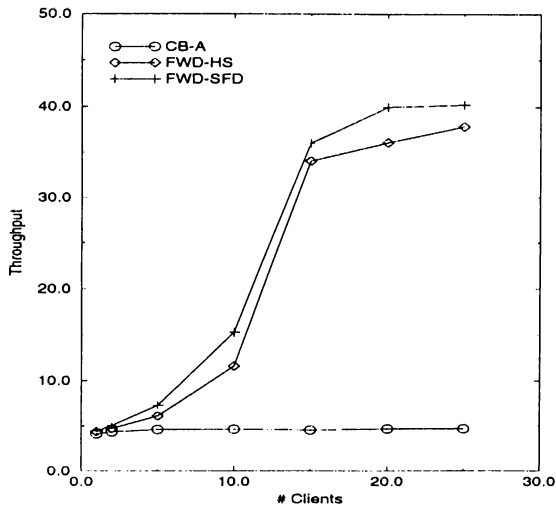


Figure 7.41. Throughput (UNIFORM, 50% Ser Buf, 5% Cli Bufs, Fast Net).

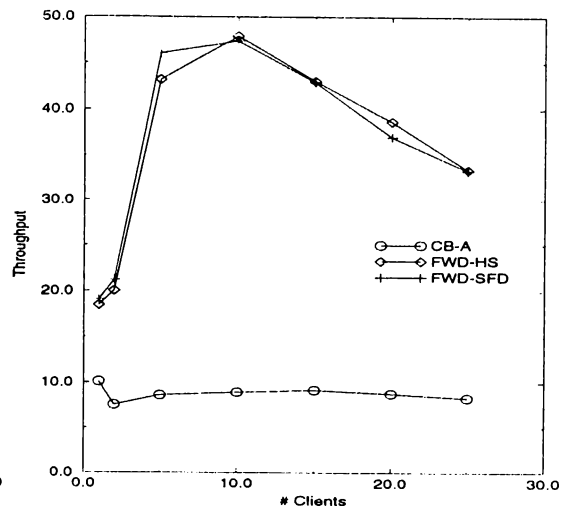


Figure 7.43. Throughput (UNIFORM, 50% Ser Buf, 25% Cli Bufs, Fast Net).

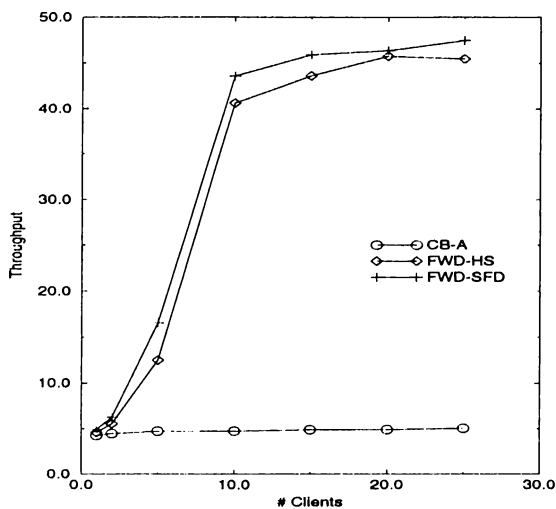


Figure 7.42. Throughput (UNIFORM, 50% Ser Buf, 10% Cli Bufs, Fast Net).

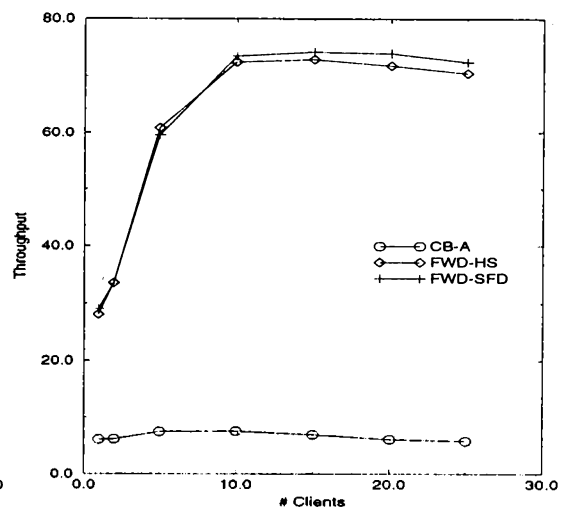


Figure 7.44. Throughput (UNIFORM, 50% Ser Buf, 50% Cli Bufs, Fast Net).

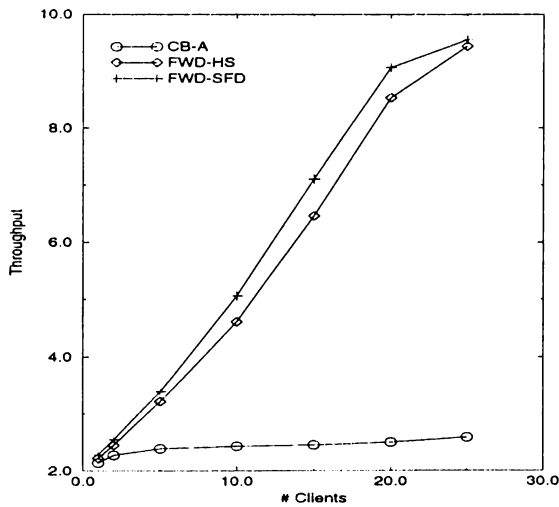


Figure 7.45. Throughput (UNIFORM, 10% Ser Buf, 10% Cli Bufs, Slow Net).

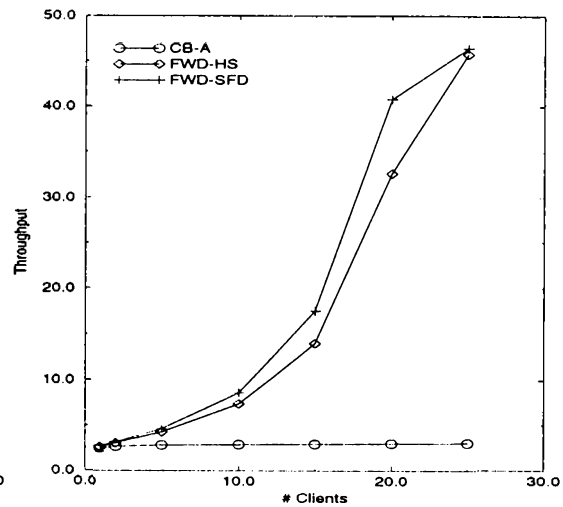


Figure 7.47. Throughput (UNIFORM, 10% Ser Buf, 10% Cli Bufs, Fast Net).

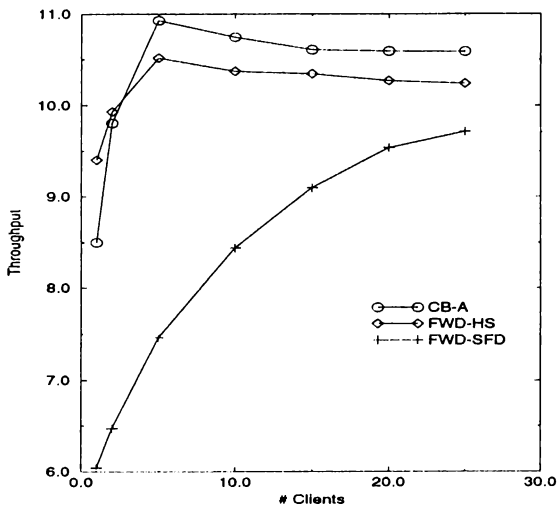


Figure 7.46. Throughput (UNIFORM, 100% Ser Buf, 10% Cli Bufs, Slow Net).

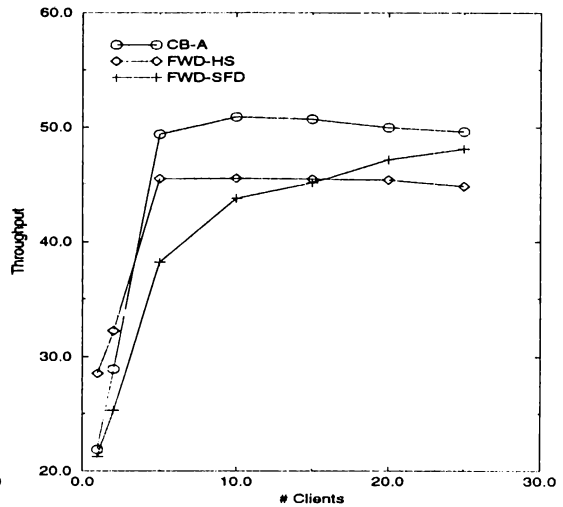


Figure 7.48. Throughput (UNIFORM, 100% Ser Buf, 10% Cli Bufs, Fast Net).

7.4 HICON Workload

In HICON workload, 80% of the accesses are directed to the same HotRegion (i.e., pages 1 to 250), and the remaining accesses are directed to the rest of the database (i.e., ColdRegion). Figures 7.49 to 7.52 show the throughput results obtained under the slow network. FWD-SFD and FWD-HS produce similar performance results and both algorithms outperform CB-A. When the buffer size becomes larger than the number of pages in the HotRegion, with CB-A the throughput of the system with one client becomes greater than the throughput of the system with two clients. This result can be explained by the fact that when there is only one client in the system the whole HotRegion is kept in the client's buffer. When one more client is added to the system, the HotRegion pages are shared among the clients (i.e., client-client replication) and the data contention is increased. However, the same situation is observed with FWD-HS and FWD-SFD when the client buffer size is equal to the 50% of the database size. When there is only one client in the system no data contention occurs and nearly 100% of the database is kept in the client's and the server's buffers, so that the total disk I/O is reduced.

Each of the three of algorithms shows similar performance characteristics under both the fast network and the slow network (for fast network results see Figures 7.53 to 7.56). When the server buffer size is small, FWD-SFD exhibits a little bit better performance than FWD-HS, as the network speed increases (see Figures 7.57 and 7.59). However, when the server buffer size is increased, because of the Dropping Sent Pages technique the performance of FWD-SFD becomes the worst, even though there exists a little bit improvement in the performance of FWD-SFD under the fast network (see Figures 7.58 and 7.60).

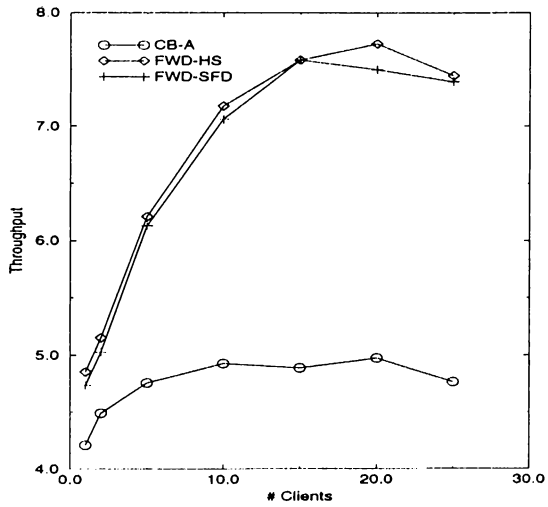


Figure 7.49. Throughput (HICON, 50% Ser Buf, 5% Cli Bufs, Slow Net).

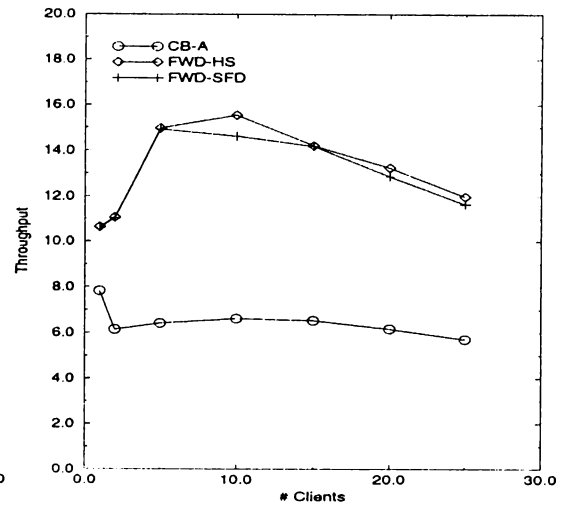


Figure 7.51. Throughput (HICON, 50% Ser Buf, 25% Cli Bufs, Slow Net).

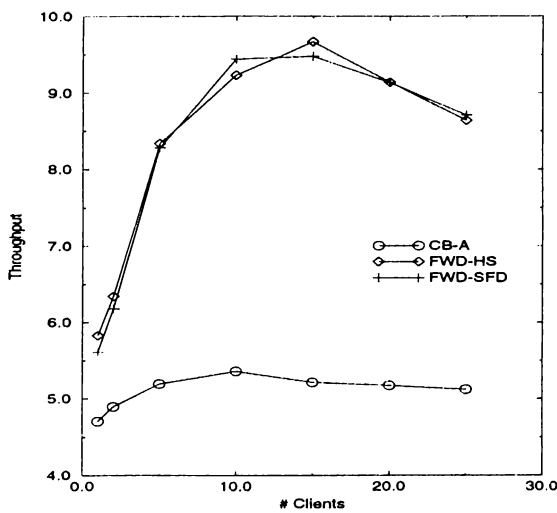


Figure 7.50. Throughput (HICON, 50% Ser Buf, 10% Cli Bufs, Slow Net).

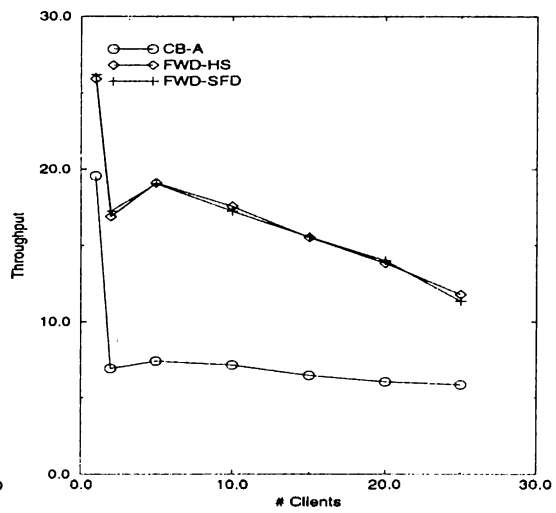


Figure 7.52. Throughput (HICON, 50% Ser Buf, 50% Cli Bufs, Slow Net).

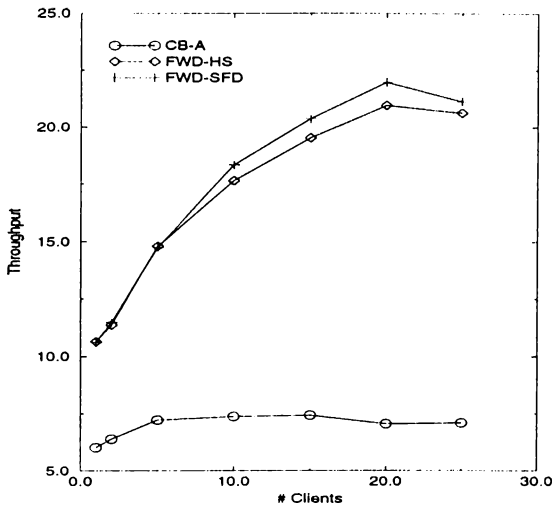


Figure 7.53. Throughput (HICON, 50% Ser Buf, 5% Cli Bufs, Fast Net).

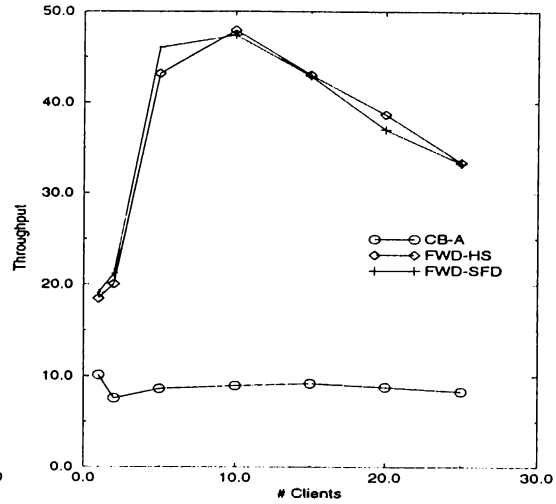


Figure 7.55. Throughput (HICON, 50% Ser Buf, 25% Cli Bufs, Fast Net).

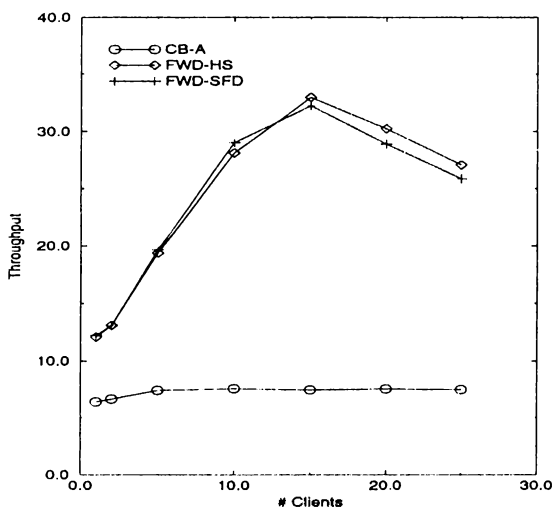


Figure 7.54. Throughput (HICON, 50% Ser Buf, 10% Cli Bufs, Fast Net).

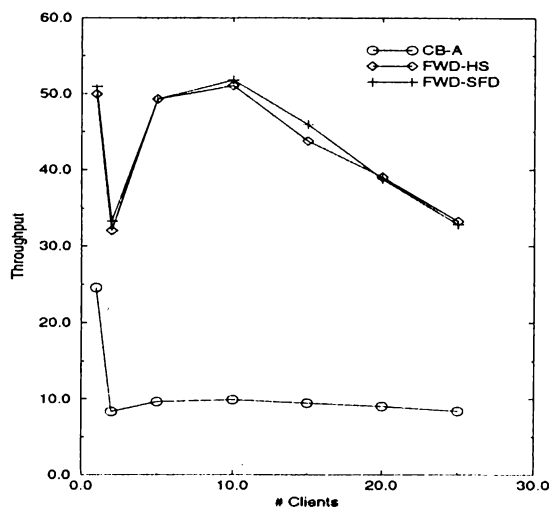


Figure 7.56. Throughput (HICON, 50% Ser Buf, 50% Cli Bufs, Fast Net).

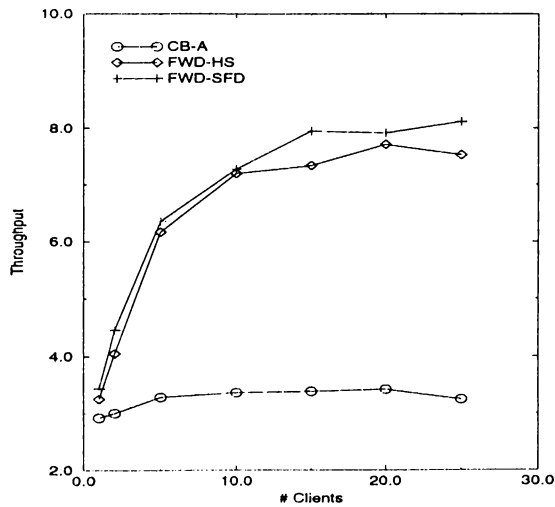


Figure 7.57. Throughput (HICON, 10% Ser Buf, 10% Cli Bufs, Slow Net).

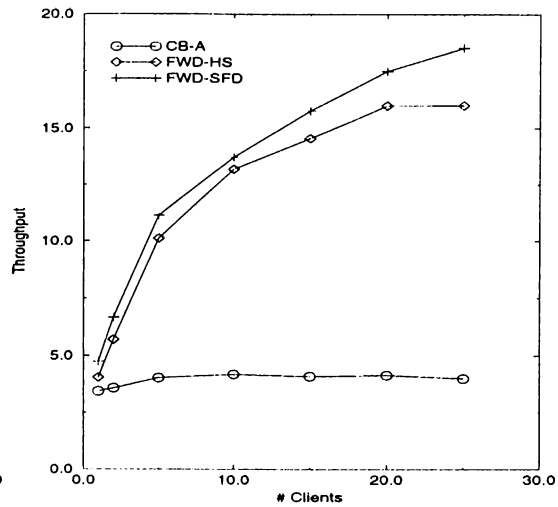


Figure 7.59. Throughput (HICON, 10% Ser Buf, 10% Cli Bufs, Fast Net).

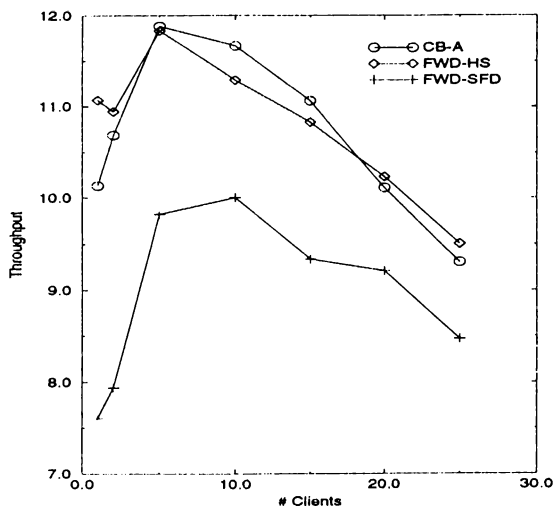


Figure 7.58. Throughput (HICON, 100% Ser Buf, 10% Cli Bufs, Slow Net).

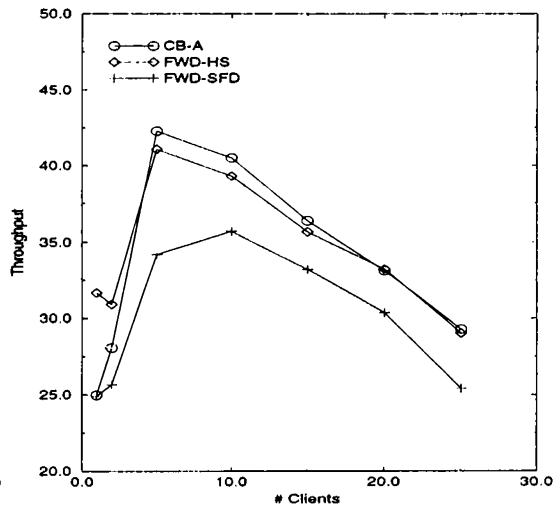


Figure 7.60. Throughput (HICON, 100% Ser Buf, 10% Cli Bufs, Fast Net).

7.5 Summary of Results

This section gives a brief summary of the performance results of the algorithms obtained with the HOTCOLD, PRIVATE, UNIFORM and HICON workloads.

Summary of HOTCOLD Results

Two main factors affect the performance of the algorithms in the HOTCOLD workload: the network speed and the disk I/O. With a slow network the performance of FWD-SFD becomes considerably worse since the message exchange requirement of the algorithm is high. However, when the network speed is high, the total number of disk I/Os determines the throughput results. Two of the algorithms (FWD-HS and FWD-SFD) decrease the amount of disk I/O while increasing the portion of the database in the global memory. Therefore, they outperform the basic algorithm CB-A. Although these two algorithms reduce the disk I/O due to the page requests, they have no effect on the disk I/O due to the dirty pages sent at commit times. FWD-SFD exhibits better performance than FWD-HS, as it exploits the clients' caches for the dropped pages reducing the disk I/O. FWD-HS receives all the dropped pages, and when its buffer is full it has to perform disk I/O for the page replacement.

When the server buffer is small, the global memory management techniques used to increase the portion of the database is really important. However when the server buffer is large (the ideal case is 100% of the database size), applying these techniques decreases the throughput as redundant operations are done in the algorithms (e.g., sending dropped pages, dropping the sent page out of server buffer, etc.). The global memory management algorithms become more applicable when the network speed is high, so that the redundant communication becomes less important.

The client buffer size is another factor that affects the performance of the algorithms. When the client buffer size increases the client cache hits also increases, as more of the HotRegion data is kept in the buffer, and the portion of the database in the global memory is also increased. Therefore, the disk I/O is decreased when the buffer size is increased. The client buffer size becomes even more important when the network speed is slow.

Summary of PRIVATE Results

Network speed affects the performance of the algorithms in the PRIVATE workload. Under a slow network CB-A provides the best performance while FWD-SFD exhibits the worst performance. However, when the network speed and the number of clients increase FWD-SFD starts to exhibit better performance. Server buffer size is another factor that affects the performance of the algorithms. When the server buffer size is small (i.e., 10% of the database size) FWD-SFD provides the best performance. However, as the server buffer size increases the throughput of the FWD-SFW decreases with respect to the CB-A and FWD-HS algorithms.

Summary of UNIFORM Results

In the UNIFORM workload, both the network speed and the server buffer size affect the performance of the algorithms. Under a slow network FWD-HS and FWD-SFD show nearly the same performance while under a fast network the performance of FWD-SFD becomes a little bit better than the performance of FWD-HS. When the server buffer size is small, as expected, the performance of FWD-SFD becomes better than that of FWD-HS. However, when the server buffer size is large CB-A exhibits the best performance and FWD-SFD the worst.

Summary of HICON Results

Network speed does not affect much the performance of the algorithms in the HICON workload. FWD-SFD and FWD-HS exhibit the similar performance characteristics, and they both outperform CB-A under both the fast network and the slow network. However the server buffer size considerably affect the performance of the algorithms. When the server buffer size is small, FWD-SFD provides a little bit better performance than FWD-HS. As the server buffer size increases CB-A and FWD-SFD start to exhibit similar performance while FWD-SFD exhibits the worst performance.

Chapter 8

CONCLUSION

There has been a great improvement in the price/performance of the workstations, and the networking capabilities in recent years. Therefore, the client-server system architecture has become a target for the new generation of database systems. One of the research fields that aims to provide high performance client-server database systems involves in the development of efficient techniques for global memory management. With today's technology, accessing data on the local memory or transferring remote data over the network is faster than accessing the data stored on the disk. Therefore, designing efficient global memory management algorithms aims to decrease the number of the disk I/Os experienced by each committed transaction.

Franklin has proposed the Forwarding technique to increase the portion of the database available in the global memory. This technique creates a global memory hierarchy for an efficient memory management. For each page request of a client, first the server's buffer is searched; if it is not found, other clients' buffers are searched, and if the page still can not be found, it is read from the disk. Franklin has provided two other memory management techniques (i.e., Hate Hints and Sending Dropped Pages) to improve the system performance. With the Hate Hints technique if the server sends a page to a client, it marks the page as hated (i.e., it marks the page as the least recently used one to replace it as soon as possible), and with the Sending Dropped Pages technique the server buffer pool is used to prevent a page to be completely dropped out of the global memory.

In this thesis two new techniques are proposed for the purpose of increasing the portion of the database available in the global memory, and thus reducing the number of I/Os per commit. The first technique, called Dropping Sent Pages, is an extension to the Hate Hints technique. With this technique, after sending the page to a client, the server just drops the page from its memory instead of marking the page as hated. The second technique, called Forwarding Sent Pages, exploits the clients' buffer that are free to receive the dropped pages, so that the only copies of the pages can be kept in the global memory.

The results of the performance experiments conducted using a simulation model have led to the following conclusions. Forwarding technique provides a significant performance improvement over the non-forwarding algorithm Callback-All (CB-A) which was used as the base algorithm in our evaluations. The proposed memory management techniques used to increase the portion of the database available in the global memory provide further improvements in the throughput performance under certain conditions. With the new techniques, the portion of the database available in the global memory increases, the total disk I/Os decreases, and as a result, the performance becomes better. On the other hand, these techniques lead to an increase in the total number of messages exchanged and thus the message volume in the system. Under fast networks this drawback is less important, as mainly disk I/Os affect the performance.

Different extensions to the proposed memory management techniques are possible. In the Sending Dropped Pages technique, the dropped pages are sent under the control of the server. Instead of communicating with the server for each dropped page, these pages can be broadcasted, and the idle clients listening the network can pick the pages. Another extension is possible for the page replacement policies of the clients and the server. The clients and the server use the LRU protocol in deciding which page should be dropped. Other techniques can also be adapted, such as, values can be assigned to each page (e.g., higher values can be given to the pages in the HotRegion), and the valuable pages are not dropped. There is only one server in the system that is investigated in our work. However, systems having more than one server can also be investigated. Instead of having one large server, employing many smaller servers can be more efficient depending on the price/performance characteristics of these systems. The whole database can be partitioned among these

servers using a particular hashing function. The servers need not be dedicated servers. Each client can also behave as a server to other clients by owning a portion of the database. While it is requesting pages from other servers, it can also serve the page requests coming from other clients. Another restriction of our model is that only one transaction can be running at a time at each client. It is also possible to implement clients as multiprocessing workstations. The performance of the proposed techniques can also be tested under these conditions. Finally, as discussed in [10], a new level of storage hierarchy can be introduced by adding local disks to the clients, and the performance of the memory management techniques can be evaluated under such an architecture.

Bibliography

- [1] B. Nitzberg, V. Lo. *Distributed shared memory: A survey of issues and algorithms*. *IEEE Computer*, 24(8):52–60, August 1991.
- [2] C. Pu, A. Leff, F. Korz, S. Chen. *Redundancy management in a symmetric distributed main-memory database*. Technical Report CUCS-014-90, Department of Computer Science, Columbia University, 1990.
- [3] C. Pu, D. Florissi, P. Soares, K. Wu, P. S. Yu. *Performance comparison of dynamic policies for remote caching*. *Concurrency: Practice and Experience*, 5(4):239–256, June 1993.
- [4] D. J. DeWitt, P. Fattersack, D. Maier, F. Velez. *A study of three alternative workstation-server architectures for object oriented database systems*. Technical Report #936, Department of Computer Science, University of Wisconsin-Madison, 1990.
- [5] M.J. Franklin. *Caching and Memory Management in Client-Server Database Systems*. Ph.D. thesis, University of Wisconsin-Madison, 1993.
- [6] K. Wilkinson, M. Neimat. *Maintaining consistency of client-cached data*. *Proceedings of the 16th VLDB Conference*, pages 122–133, Brisbane, Australia, 1990.
- [7] M. J. Carey, M. Livny. *Conflict detection tradeoffs for replicated data*. *ACM Transactions on Database Systems*, 16(4):703–746, December 1991.
- [8] M. J. Carey, M. J. Franklin, M. Livny, E. Shekita. *Data caching tradeoffs in client-server DBMS architectures*. Technical Report #994 Computer Sciences Department, University of Wisconsin-Madison, January 1991.

- [9] M. J. Franklin, M. J. Carey. *Client-server caching revisited*. Technical Report #1089, Computer Sciences Department, University of Wisconsin-Madison, May 1992.
- [10] M. Livny, M. J. Franklin, M. J. Carey. *Local disk caching for client-server database systems*. *Proceedings of the 19th VLDB Conference*, pages 641-654, Dublin, Ireland, 1993.
- [11] M. J. Franklin, M. J. Carey, M. Livny. *Global memory management in client-server DBMS architectures*. Technical Report #1094, Computer Sciences Department, University of Wisconsin-Madison, June 1992.
- [12] M. J. Franklin, M. J. Carey, M. Zaharioudakis. *Fine-grained sharing in a page server OODBMS*. Technical Report #1224, Computer Sciences Department, University of Wisconsin-Madison, April 1994.
- [13] M. Stumm, S. Zhou. *Algorithms implementing distributed shared memory*. *IEEE Computer*, 23(5):54-64, May 1990.
- [14] R. Hagmann, D. Ferrari. *Performance analysis of several back-end database architectures*. *ACM Transactions on Database Systems*, 11(1):1-26, March 1986.
- [15] E. Rahm. *Performance evaluation of extended storage architectures for transaction processing*. *Proceedings of ACM SIGMOD Conference*, pages 308-317, 1992.
- [16] H. Schwetman. *CSIM Reference Manual*. Microelectronics and Computer Corporation, 1986.
- [17] H. Schwetman. *CSIM: A C-based process-oriented simulation language*. *Proceedings of Winter Simulation Conference*, pages 387-396, 1986.
- [18] M. Stonebraker. *Architecture of future database systems*. *IEEE Data Engineering*, 13(4):37-65, December 1990.
- [19] Sun Microsystems. *Network Programming Guide*.
- [20] U. Çetintemel, A. Çınar, G. D. Tunali, Ö. Ulusoy. *A simulation model for client-server database management systems*. *Proceedings of the Ninth International Symposium on Computer and Information Sciences*, pages 81-88, November 1994.

- [21] Y. Wang, L. A. Rowe. *Cache consistency and concurrency control in a client/server DBMS. Proceedings of ACM SIGMOD Conference*, pages 367–376, 1991.