Techniques for Update Handling in the Enhanced Client-Server DBMS

Alex Delis, *Member*, *IEEE*, and Nick Roussopoulos

Abstract—The Client-Server computing paradigm has significantly influenced the way modern Database Management Systems are designed and built. In such systems, clients maintain data pages in their main-memory caches, originating from the server's database. The Enhanced Client-Server architecture takes advantage of all the available client resources, including their long-term memory. Clients can cache server data into their own disk units if these data are part of their operational spaces. However, when updates occur at the server, a number of clients may need to not only be notified about these changes, but also obtain portions of the updates as well. In this paper, we examine the problem of managing server imposed updates that affect data cached on client disk managers. We propose a number of server update propagation techniques in the context of the Enhanced Client-Server DBMS architecture, and examine the performance of these strategies through detailed simulation experiments. In addition, we study how the various settings of the network affect the performance of these policies.

Index Terms—Client-server DBMSs, update propagation policies, cluster database computing, push servers, modeling of client-server DBMSs, simulation and performance analysis.

1 Introduction

Organizations and companies are deploying database servers at an ever increasing rate in order to meet their basic business requirements. The demand for retrieval and manipulation of very large amounts of data is also increasing rapidly [1], [51], [12]. This trend calls for high throughput database systems and scalable architectures that demonstrate excellent performance characteristics. Previous efforts in the realization of high performance database computing include database machines and multiprocessor databases [18], [6], [8]. Although these efforts offered solutions in the area of database performance, they demanded excessive costs since they required specialized hardware and software.

In recent years, the Client-Server computing paradigm [48] has gained wide-spread acceptance and has been used extensively in the development of contemporary computing systems. Technological advances coupled with reduced pricing in hardware have created a new reality [46]. More specifically, we have experienced the wide availability of inexpensive workstations and powerful PCs, the introduction of large, fast, and reliable disk units, as well as the appearance of fast local area networks [37], [25], [4]. By taking advantage of this infrastructure, the Client-Server paradigm attempts to satisfy the requirements for both high performance and scalability [14]. The successful fulfillment of the last two requirements can lead us to a new era of intense data sharing and network-centric database computing [1]. To this end, Client-Server Database Systems (CS-DBMSs) have been introduced in order to accommodate data sharing and improve performance.

The fundamental concept in CS-DBMSs is that a dedicated machine runs a DBMS and maintains a main centralized database (DBMS-Server) [17], [41], [15], [28]. The users of the system access the database through either their workstations or PCs, via a local area network. The interaction between workstations/PCs and server is achieved by the underlying operating systems and their interprocess communication abstraction mechanisms, such as Remote Procedure Calls or sockets [7], [48]. The workstations typically run user-interface programs and direct all database inquiries and/or updates to the DBMS-Server. In this way, they become the server's clients. This clustered configuration, which often runs on a Network of Workstations (NOW), has been termed Standard Client-Server DBMS (SCS) [14]. Although the environment in SCS is distributed, the DBMS is centralized and, therefore, transaction handling is easier than in distributed databases [11]. The server attempts to satisfy every incoming client request by first creating and then executing a thread [27]. Threads execute concurrent tasks as different streams of control in the context of a multithreaded database server process [23]. Both commercial products [9], [26], [3], [29], [31], [49] and research prototypes [53], [52], [19], [22], [40], [38] use some variation of this basic Client-Server model.

The Enhanced Client-Server DBMS (ECS-DBMS) [15] offloads disk accesses from the server. This is achieved by having clients run a limited DBMS in terms of concurrency and by caching results of server queries to the client disk managers. The rationale for this type of disk caching is to boost clients' performance by avoiding potentially highly-loaded servers. In this paper, we study the performance of update propagation techniques for the ECS type of Client-Server DBMS architecture as the number of participating

- *A. Delis is with the Department of Computer and Information Science, Polytechnic University, Brooklyn, NY 11201. E-mail: ad@naxos.poly.edu.*
- *N. Roussopoulos is with the Computer Science Department and Institute for Advanced Computer Science (UMIACS), University of Maryland, College Park, MD 20742. E-mail: nick@cs.umd.edu.*

Manuscript received 27 Feb. 1995; revised 16 Oct. 1997.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104909.

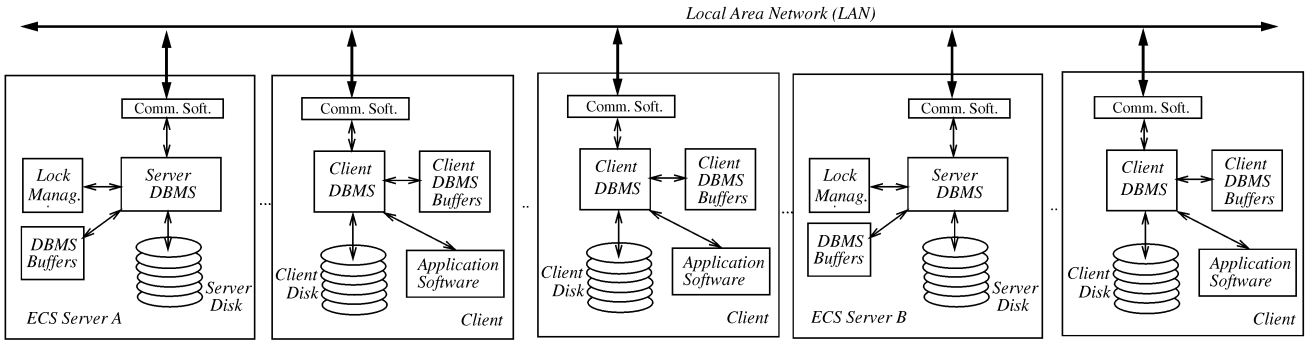

Fig. 1. Servers and clients in the ECS architecture.

clients in the cluster increases. A number of update propagation techniques, whose features range from pure push-servers to self-contained pull-clients, are presented and their performance is studied under a number of workloads through simulation experiments. The simple broadcasting strategy for server updates (push-server) gives better performance rates over the other policies in a number of occasions, while the on-demand update propagation strategy furnishes better results under the condition that none of the server resources reaches full utilization.

The paper is organized as follows: Section 2 gives an overview of the ECS Client-Server DBMS. In Section 3, we state the problem and compare this work with prior related studies. Section 4 suggests a number of policies, and discusses policy characteristics and overheads. Section 5 briefly describes the simulation models used and discusses the system parameters. In Section 6, we present the evaluation methodology and our experimental results. Conclusions can be found in the last section.

2 Features of the Enhanced Client-Server DBMS

The Standard Client-Server (SCS) model originated in engineering applications where data are mostly processed in clients with powerful CPUs [46], [26], [3]. In these applications, centralized repositories are used to maintain global data consistency with check in and check out protocols. The Standard Client-Server configuration uses the network as the means to either send messages or ship query results from the server to clients. Clients run presentation managers and application programs locally while they direct data requests to the database servers. Extensive database operations initiated by a number of concurrent clients may generate heavy loads and significant processing delays at the server nodes of the architecture [14].

The Enhanced Client-Server architecture (ECS) [15] offers relief to database servers by utilizing both CPU cycles and I/O capabilities of its clients. The functionality of the clients is "enhanced" to a simplified single-user DBMS. The single-user DBMS in discussion takes advantage of the often voluminous long-term memory spaces available at the client level. In this setting, clients may cache data of their interest (i.e., query results) into their local disk managers for future re-use. Enforcement of data consistency throughout the architecture can be undertaken by the servers that are the primary sites of data. Hence, servers become the caretakers for updates and their propagation to pertinent clients. Fig. 1 shows the general organization of this architecture with two server nodes and three clients. In the remainder of this paper, we will consider issues in the context of a single *cluster*. A cluster consists of one server and a varying number of clients attached to it. Such a cluster provides the basis for our discussion.

By caching query results over time, a client creates a subset of the server database on its own disk unit. A client essentially maintains a partial replica of the server database which is of interest to the client's application(s). Later on, a user can integrate into her client database individual data not accessible and of no interest to others. There are two major advantages for this type of disk caching: First, repeated requests for the same server data are eliminated and, second, system performance is boosted as clients can access local data copies. Nonetheless, in the presence of updates, the system needs to ensure proper propagation of either new objects or modified elements to the appropriate clients.

Every time a client decides to cache the results of a query into its own disk, it creates a "new-local" relation. This newly created entity is derived from the server tables and it must be affiliated with these tables. Each such relationship between client cached data and server relations is designated by a client binding. Bindings are described by templates consisting of three elements: the participating server relation(s), the applicable condition(s) used to extract data from the server relations (or predicates), and a timestamp. Conditions essentially play the role of a filtering mechanism and designate which server tuples are of interest to a specific client. The timestamp of a binding indicates the *last* time the data of a client has been refreshed with updates that have taken place at the server. Since that *last* time, other modifications may have taken place and they have possibly caused client data to become inconsistent.

Updates are, in general, directed for execution to the server which is the primary site. Pages to be modified are read into main memory, updated, and flushed back to the server disk unit. In ECS, every server relation is associated with an update propagation log which consists of time-stamped inserted tuples and timestamped qualifying conditions (predicates) for deleted tuples. Only updated (committed) tuples are recorded in these logs. The number of bytes written to the log per update is generally much smaller than the size of the pages read into main memory [47]. Fig. 2

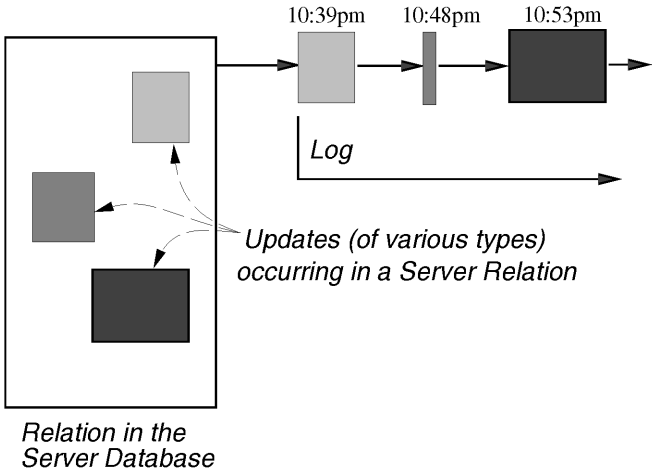

Fig. 2. Server relation with its incremental log.

shows a server relation and its accompanied log with three modifications that have happened at various times. For instance, the operation that took place at 10:48 p.m. was a deletion of records from the server relation. As such, the log only maintains the description of this logical operation.

ECS client query processing against local data is preceded by a request for an incremental update of this data. The server is required to look up the portion(s) of the query-involved relation logs that maintain entries with timestamps greater than the one seen by the submitting client so far. The look-up process in discussion may be carried out once the binding template for the requesting client is available.

The set of all binding templates can be either stored at the server's catalog (in the form of a binding directory) or collectively maintained by the individual clients. In the former case, the server maintains all the information regarding the caching status of the clients (i.e., which client has "seen" what portions of the server relation logs and up to what time). Naturally, these statistics multiply quickly as the number of clients attached to a server increases. The second alternative releases the server's DBMS from keeping track of a large number of cached data subsets and the different update states. However, if the distributed option is taken, then a client's request should be accompanied with the proper binding template.

The binding template enables the server to perform the correct data filtering. Only relevant fractions (i.e., increments) of the modifications (relation update logs) are propagated to the client's site. A set of algorithms that carry out such operations are based on the incremental access methods for relational systems [39]. These algorithms look up the update logs of the server and transmit differential files. Shipment of short differential files proves to be valuable in settings where the bandwidth of the network is limited.

The concurrent processing of all updates and query/log operations is carried out by the Server DBMS, as Fig. 1 indicates. In order to maintain consistency, data pages are accessed through a standard locking protocol such as the strict-2ϕ locking protocol [23]. The lock tables assist the server DBMS in carrying out this protocol. Once a lock has been acquired, the disk manager can schedule a page transfer into the server DBMS buffer area for processing. We assume that the server DBMS buffer area is much smaller than the size of the database. We also assume that the client DBMS buffers are smaller than the size of the cached data on the client disk.

3 Problem Formulation and Related Work

The fundamental question addressed in this paper could be summarized as follows: Given an ECS-DBMS configuration and a server committed update, what are the alternatives in propagating/pushing the results of this operation to interested clients? The cached disk-resident data of these clients may become inconsistent after the update commits.

Although the issue in its general framework is not new, it has been examined under different contexts in the past. Alonso et al. [1] examine a similar problem in information systems. They relax the consistency of the client disk cached data and study the performance of Client-Server information systems using quasi-copies of data. Carey et al. [10] examine the performance of five algorithms that maintain consistency of cached data in CS-DBMS. However, the important assumption of this work is that client data are maintained in cache memory and they are not disk resident. Similar work was carried out by Wang and Rowe [52]. Franklin et al. examine the performance of various page-oriented caching techniques [21]. Mohan and Narang [33] propose a recovery algorithm for a Client-Server database environment where server and client clocks do not need to be synchronized. There is also a large amount of work done in the areas of cache coherence algorithms [2] and distributed shared main memories [36]. The major problems examined in the last two areas could be characterized as isomorphic to our problems with the ECS configuration. Work in cache coherence examines how it is possible to maintain memory caches up-to-date with the contents of the main memory and increase overall system performance. In distributed shared main memories, the common channel (or bus) that connects the memories is a serialization element and thus, a potential bottleneck. In the latter, the main issue is how to attach multiple processors to a large group of distributed shared main memories without creating bus congestion.

When the problem is examined in the context of the Enhanced Client-Server DBMS configuration, there are a number of elements that impose new constraints. These constraints stem from the fact that databases work predominantly with disk resident data and that the CPU time to process database operations is not negligible in both server and clients. Other questions that can be examined in this setting are:

1) What is the achieved relative performance of the various propagation alternatives?
2) How do these strategies scale up in the presence of many clients (more than 30-40)?
3) Is there any gain in employing an incremental and on-demand propagation strategy?
4) In a rare-update environment, there is no data inconsistency and clients work off their copies providing a system with almost linearly scalable performance. As

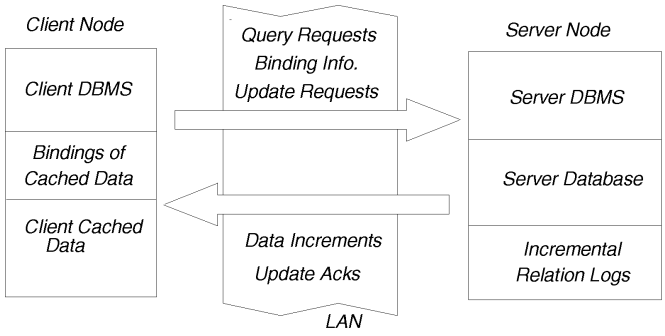

Fig. 3. Logical diagram for the ODM policy.

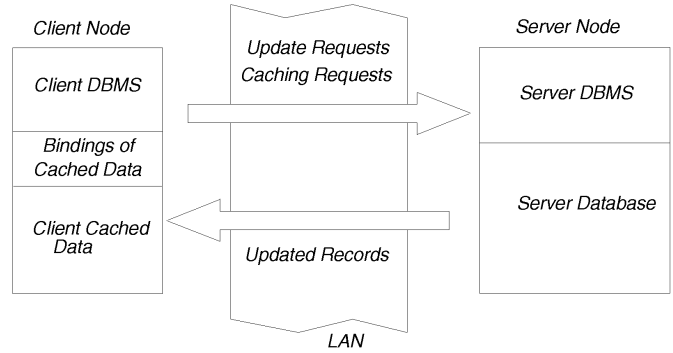

Fig. 4. Logical diagram for the BNC policy.

updates increase and their operational areas on the database become larger, what is the overhead that needs to be paid by both the clients and the server to offer timely update propagation?

Some work indirectly related to this study was done in the areas of multiprocessor DBMS architectures [6], fully distributed DBMSs [35], [5], [44], and distributed systems [50], [30], [42].

4 Strategies for Update Handling

In this section, we introduce five possible strategies for ECS data propagation and talk about their design rationale and supporting mechanisms.

On-Demand Strategy (ODM): This policy has been used in the model of the Enhanced Client-Server model outlined in Section 2. The key idea is that the server does not need to do any bookkeeping in terms of data bindings (e.g., a distributed alternative for the management of the bindings is used). This implies that if a client wants to materialize a query, it has to present the server with its binding/caching information as well as the query itself. In this way, the server is capable of identifying the data space of interest to every individual client request and commences the appropriate actions to service this request. The binding information works as the filter that determines the portions of the server relation logs that need to be forwarded to clients.

Fig. 3 presents the logical architecture of the configuration based on the ODM strategy. In general, there is more than one client attached to the LAN. Query messages, binding information, and update requests are directed from the clients through the network to the server. Data increments and update commit acknowledgments are forwarded from the server to the clients.

The second group of strategies is built around the idea of broadcasting (pushing) server data modifications to all clients in the cluster as soon as an update commits [34]. The rationale is that if the updated tuples are already in main memory, then we could avoid subsequent data rereading from the disk when the need for update propagation arises. Therefore, some time could be saved and logs become unnecessary. Depending on how the volume of the data broadcast is decided, two alternative policies are introduced: Broadcasting with No Catalog bindings and Broadcasting With Catalog bindings.

- *Broadcasting with No Catalog bindings* (BNC): This policy pushes just committed updates to all clients in a cluster indiscriminately. The CS configuration that operates under this policy requires no additional server overhead in terms of functionality. The server avoids look-up log operations and computations needed to determine the destination of the updates. As soon as a write operation commits, the server establishes a communication channel with every client (point-to-point). Through this channel, updated tuples (data) are pushed to workstations. After the transfer is finished, the server closes the communication channel indicating the completion of the broadcast. When a client receives the changes, it suspends any on-going work and determines if the modifications broadcast affect its operational locale in any way. The latter can be easily established with the client catalog information at hand. If a client has to abort the current job, it then flushes (commits) the newly arrived changes into its disk and restarts the just aborted query. Fig. 4 shows the logical structure of this strategy. The network traffic consists of update requests and updated records. Queries are executed solely at the clients, without any server interaction.
- *Broadcasting With Catalog bindings* (BWC): This policy extends the previous scheme by trying to limit the amount of data transported over the network (Fig. 5). This is done by reducing the volume of data based on server-maintained binding information. A directory of binding information for each client has to be maintained in the server DBMS system area (*Binding Directory* in Fig. 5). This directory states, in terms of predicates or bindings, the specific areas of the database that each client has cached into its disk. Every time an update job commits, the server opens a communication channel with a specific client only if its binding calls for it. In addition, when this channel is established, only a portion of the updated tuples needs to travel over the network—those pertinent to the client's locale. Any query executing at the client site during the broadcasting is aborted and can be restarted after the incoming modifications have been committed into the client disk manager. The directory of the bindings could be maintained in main memory. However, when the number of clients increases, such

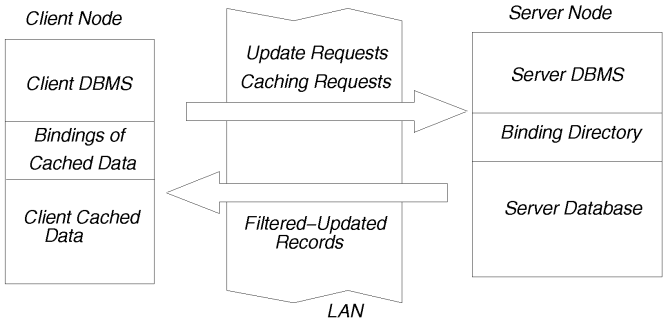

Fig. 5. Logical diagram for the BWC policy.

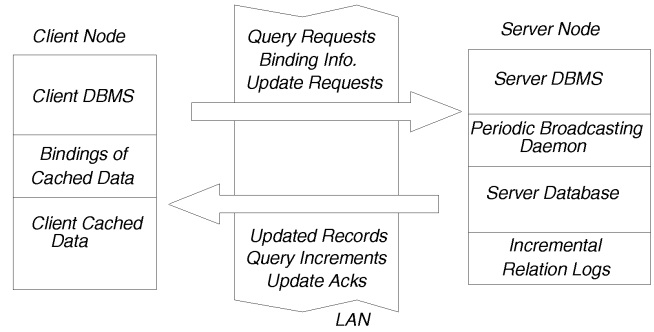

Fig. 6. Logical diagram for the PNC policy.

an assumption is not realistic and the binding directory has to be maintained in the long-term memory.

Two additional propagation policies are proposed by combining the concepts described so far and by incorporating the idea of periodic update broadcasting. In periodic update broadcasting, logs are used as the main tool to record the "net changes." Here, client-originated queries are handled in a manner similar to the one described in the ODM configuration. The additional feature is that, at regular intervals, the server is interrupted by a daemon. The daemon essentially collects all the changes "not seen" by the clients yet and initiates their propagation. It is anticipated that this type of data "prefetching" may offer gains on possible server idle time periods. During these periods certain useful propagation work may take place. Nevertheless, it is expected that under stringent job submission times (short think times) the periodic propagation will suffer.

As soon as the server daemon reads the "not yet seen" portions of the log into the main-memory buffers, it can push them to the various clients. This transmission of log portions can be carried out by either a naive or a discriminatory broadcasting strategy. The former strategy results in *Periodic broadcasting with No Catalog bindings* (PNC) and the latter in *Periodic broadcasting With Catalog bindings* (PWC). The qualitative difference of the two strategies above is the same as that between BNC and BWC. Periodic broadcasting With relation Catalog bindings (PWC) attempts to limit the volume of data transferred over the local area network using server catalog information about the operational areas of every client. When modifications broadcast are received, clients operate in ways similar to the ones outlined earlier (e.g., interrupting any on-going operation, flushing received data into the local disk, etc.) Figs. 6 and 7 show functional diagrams for the PNC and PWC policies, respectively.

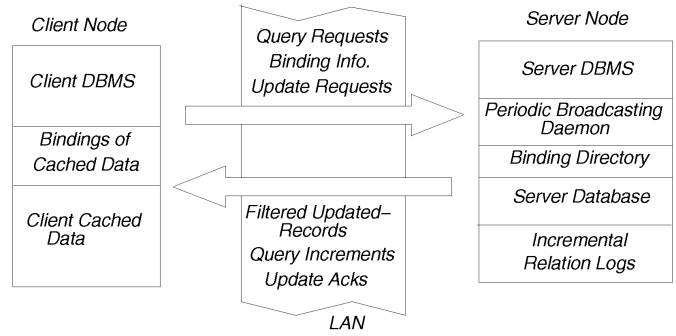

Fig. 7. Logical diagram for the PWC policy.

5 Simulation Models

We have developed software packages corresponding to the functionality of the five update propagation policies. Fig. 8 shows the high-level models used to derive our packages; models for a server, the communication medium, and one client only are shown.

The server model maintains all the key processing elements of a database system [24]. The top layer of the model termed **`Server Database Manager`** parses the incoming messages, carries out request decomposition, and places jobs in the queue of the database scheduler. The scheduler in conjunction with the **`Concurrency Control Manager`** determines when a job has to be admitted for processing by the database engine (internals of the server). The **`Concurrency Control Manager`** overlooks the maximum allowed number of jobs scheduled for processing, manages the locking mechanism, maintains the wait-for graph of the active jobs, detects deadlocks, and, finally, releases the necessary locks upon a job's completion. An externally imposed multiprogramming degree is set in order to avoid job thrashing and remains constant throughout the execution of the client requests.

The **`Buffer, Disk & Log Manager`** controls the allocation of main-memory buffer space (equal segments are allotted to all active jobs) and manages the disk unit(s). We assume that the buffers can hold only a portion of the server database. This manager is also responsible for the incremental log operations as they were discussed in Section 4. The **`Resource Manager`** handles all the available resources at the server site, namely, the CPU, the pool of the main-memory buffers, the disk units, and the network interface. We assume no contention in both system and memory bus. Resources are allocated to the various managers as soon as they become available. While server requests are being processed, they "consume" the resources of the server such as CPU, disk I/O, buffer space etc. Time spent on the server resources for the processing of requests is recorded by the simulation software for reporting purposes. Table 1 shows settings for system features and penalties involved in the operation of the database managers.

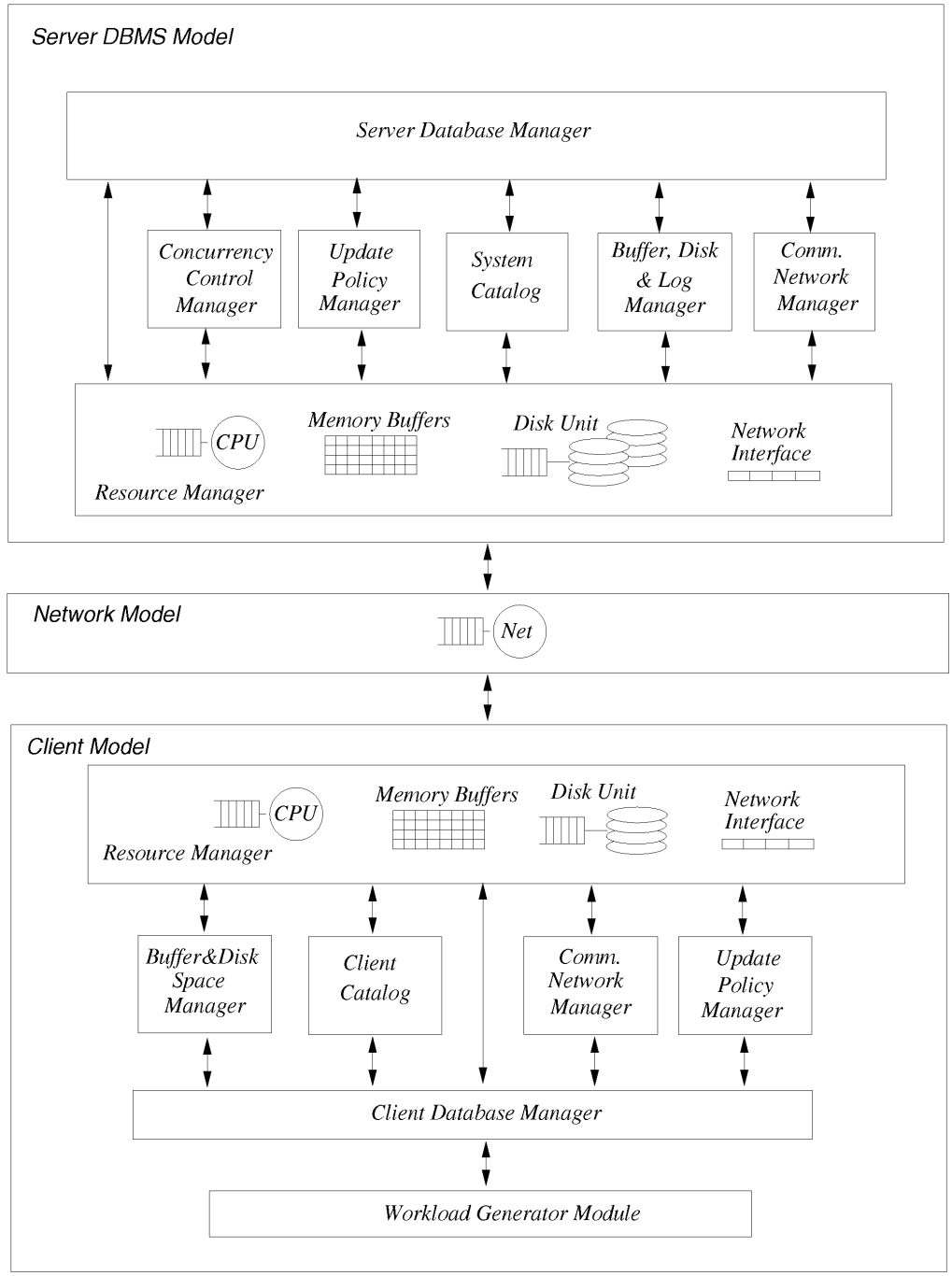

Fig. 8. Simulation models for server, communication network, and client(s).

The **Communication Network Managers** at both clients and server are responsible for the processing of the critical path of the remote process calls (RPCs) [7], [43]. The critical path is the set of instructions that has to be executed for every remote procedure call before the database manager traps to the operating system kernel. During this processing, data are segmented to packets, converted to the appropriate format (marshaling), and augmented with header fields such as destination addresses. Packets placed at the network interface are ready for transmission. At the receiving end, the corresponding **Communication Network Manager** follows the reverse process: Packets are stripped off their header addresses, unmarshalled, and used to assemble the received message and/or data. Schroeder and Burrows [43] have shown that only a quarter of the total time needed to transfer a packet between two sites is spent on the Ethernet transmission. This result suggests that a serious amount of time is spent by both the dispatching and receiving nodes before a packet is attached to the network interface. In [13], a linear model is proposed to estimate such overheads; it consists of a fixed overhead per message plus penalties due to the size of the data passed through the network. Both *msg_const* and *msg_variable* (Table 2) account for these extra CPU penalties, which take place at the dispatching and receiving network interfaces. In general, these overheads increase linearly with the size of transferred data [32].

TABLE 1
PARAMETERS FOR THE DATABASE MANAGERS

DBMS Parameters	Explanation	Value
srv_cpu_mips	Server CPU Speed	41 MIPS
srv_disk_acc_tm	Avg. Disk Page Access Time	12 msec
srv_main_mem	Size of Server Buffer	2000 Pages
mpl	Server Mutliprogr. Degree	12
get_lock	Instrs to Get a Lock	1500 ins
rel_lock	Instrs to Release a Lock	500 ins
dd_search	Deadlock Detection Phase	25000 ins
kill_time	Time to Kill a Job	80 msec
read_page	Instrs to Read a Page	6500 ins
write_page	Instrs to Write a Page	8000 ins
inst_sel	Instrs to Select from a Page	10000 ins
inst_prj	Instrs to Select from a Page	11000 ins
inst_join	Instrs to Join a Page	29000 ins
inst_mod	Instrs to Modify a Page	12500 ins
inst_log	Instrs to Process a Log Page	5000 ins
inst_ism	Instrs to Log a Page in Disk	6000 ins
interval_tm	Periodic Broadcasting Interval	5 sec
cl_cpu_mips	Client CPU Speed	20 MIPS
cl_disk_acc_tm	Avg. Client Page Access Time	16 msec
cl_main_memory	Size of Client Buffer	500 Pages
inst_br	Instrs to Broadcast a Page	8000 ins
inst_br_up	Instrs to Process Broadcasted Updates at Clients	40000 ins
dir_cond	Avg. Pages Accessed Per Directory Search	2 Pages
cpu_dir_page	Avg. Time to Process a Directory Page	15 msec

TABLE 2
NETWORK RELATED PARAMETERS

Net Parameters	Explanation	Value
msg_const	Initial Cost Per Message	10000 ins
msg_variable	Overhead Per Packet	2000 ins
pckt_sz	Size of a Packet	1024 Bytes
net_rate	Network Rate	10 Mbits/sec
msg_length	Avg. Size of Messages	400 Bytes
msg_odm	Avg. Size of ODM Messages	2048 Bytes

The structure of the network model consists of a queue and the network resource (**Net**). Messages, transaction requests, and data transfers between clients and the server are routed through the network. Delays due to high contention of the communication medium may occur when the number of attached clients is increased. Requests appearing on the network interfaces of all sites receive top scheduling priority and preempt the CPU from any ongoing work.

Client models essentially reflect the need for the long-term caching of server-originated data. These data become part of the local disk space before any local query materialization commences. The client sites are responsible for the creation of the transactions that the architecture will have to complete. This task is delegated to the **Workload Generator Module**. Client sites consist of a simplified database manager that helps implement the designated update propagation policy and carries out normal transaction processing with the use of main-memory buffers and disk space. The **Client Catalog**, **Update Policy**, and **Buffer & Disk Space Managers** coordinated by the **Client Database Manager** control the way all locally available resources are used. The resources in discussion are the CPU, the existing buffer space, the communication interface, and the local disk unit; they are all integrated in the **Resource Manager**. The local disk unit is predominantly used for intertransactional long-term caching. Due to restricted buffer space available, join operations may require a number of repetitive disk accesses to the same pages [45]. We also assume that there is one user per client node.

TABLE 3
DATA RELATED PARAMETERS

Data Parameters	Explanation	Value
pg_sz	Page Size	2KBytes
Rel_Size	Relation Size	1000 pages
Num_of_Res	Number of Relations	8
α_{Rel_i}	Portion of Server-Relations Cached on Clients	0.3
wr_log_fract	Fraction of an Update into the Relation-Log	0.10

Table 3 shows the parameters related to data used in the experimentation. Only portions of the server relations are initially cached in the client site. In particular, the α_{Rel_i} parameter indicates this fraction. As Section 2 indicates, only a fraction of the server updated pages have to be recorded at the incremental logs; this fraction is represented by the *wr_log_fract* parameter [47]. The number and size of used relations, as well as the size of pages, are also shown in Table 3.

In the BNC policy, queries are processed exclusively on the clients while updates are directed to the server. After the server updates commit, the newly modified records are forwarded to all clients that participate in the cluster. If a client receives records while processing a query, then it has to interrupt its ongoing work in order to service the incoming updates. These updates have to be filtered against the locally available binding information. The purpose of this step is twofold: first, the relevance of the incoming data against the client bindings is determined and second, if pages already in the client query buffer are affected by the new arrivals, the current query has to be aborted and restarted. The above processing is represented by the client CPU overhead *inst_br_up*. The **Workload Generator Module** resubmits a request if necessary.

The **Update Propagation Manager** of the BWC server needs to retrieve binding information from the system catalog. The portion of the catalog that maintains bindings cannot be generally maintained in main memory. Therefore, a number of appropriate disk pages have to be carried out in order to access the appropriate bindings. This information is used in conjunction with the newly in-memory updated tuples so that the server determines the tuple portions suitable to each client. In the BWC policy, this penalty is expressed in terms of CPU processing with a *cpu_dir_page* overhead. CPU Penalties due to client processing for pushed data is designated by the *inst_br_up* factor.

In the PNC/PWC policies, the server **Update Policy Manager** initiates reading of the unread incremental log portions. The server charges CPU log processing (*inst_log*) time required for the retrieved log data. In PWC, we need to retrieve not only the unread portions of the logs, but also the binding information from the server catalog in order to decide the appropriate log portions for each client. Similarly to BWC, every page derived from the binding directory is charged with *cpu_dir_page* time for CPU processing.

The five simulation packages were written in C and their sizes vary between 5.3k and 6.1k lines of source code. They support concurrent job operations, automatic deadlock detection at the server, and interruption of processing at the clients, as discussed above. The run-time for each of our experiments requires approximately 26 hours of CPU time on a Sun-SPARCstation 20.

6 Experimental Results

This section briefly discusses our evaluation methodology and presents the main results of our experimentation. Two elementary workloads are used in the process; they constitute the basis for our comparison. We also discuss the role of key parameters in our analysis.

6.1 Workloads and Measurement Methodology

The means to create the client data patterns of access is that of *job streams*. A job is either a query or an update. A *job stream* is a sequence of jobs made up by mixing queries and updates in a predefined proportion. In the two extreme cases, we can have either query or update-only streams. Every client is assigned to execute such a stream. Utilizing the varying query / update ratios feature that our simulators demonstrate, we conduct two families of experiments:

1) Those with Constant number of Update jobs (CU), where a constant number of four clients submit update-only streams, and the remaining clients submit query-only streams. This setting simulates stock market environments or generally environments with few writers and many readers.
2) Those with Variable Update jobs (VU) where each stream is a combination of both queries and updates. Updates constitute 10 percent of all the jobs and are uniformly distributed over the queries of each stream, simulating traditional database environments.

Queries consist of relational operations that manipulate up to 10 percent of the pages of the server relation(s). Thus, two families of experiments were carried out using job streams that were of CU and VU stream types. The same streams were submitted for all update propagation strategies. The workload definition has been enhanced to reflect the incorporation of CPU network processing and accessing catalog pages. The objectives of the experiments are to:

- Examine how the various update propagation techniques behave under these workloads.
- Identify important parameters and study their impact on the different strategies.

Naturally, the values of the system parameters are highly implementation-dependent, but our objective is to come up with relative estimates. In the simulations, we vary two parameters: the number of participating clients from five to 100 (x axis of the graphs) and the update page selectivity from 2 percent to 8 percent. The simulators create streams by randomly selecting jobs from sets of query and update templates. The page update selectivity remains the same throughout all the modifications of the same job stream.

The main performance criterion for our evaluation is the overall average job throughput. The average throughput is measured in jobs per minute (JPM); this metric is projected on the y axis of our graphs. The number of participating jobs per stream was selected to be long enough—135 jobs per stream—to guarantee satisfactory standard deviation for our experiments. The utilized stream length resulted in standard deviation ranging between 0.1 percent and 2.6 percent, which is considered satisfactory [20].

Initially, client think time is set to zero in order to test the various update propagation strategies under stringent conditions. In our experiments, the clients have cached the data of their interest in their respective disk units before experimentation commences. The size of the cached data in every client is much smaller than that of the server database (e.g., about a third).

6.2 CU Experiment

Fig. 9 shows the performance results of the five policies for 2% update jobs in the CU workload as we increase the number of clients attached to a server. The number of update streams remains at four (4) throughout the experiment. The key observations are:

- BNC surprisingly performs better than ODM. One would expect that the on-demand strategy should give the best rates. In the ODM strategy, there are incremental log pages that have to be first written into the disk and subsequently read on behalf of the various clients. In the context of the BNC policy, no such reading / writing takes place. Updated tuples from the main memory buffers (just before or after the transaction commit) are put forward to the network interface. The BNC does not only require server CPU time but also increases the network utilization as it employs point-to-point communication. The ODM policy poses CPU processing requirements for incremental log but, as the sizes of data increments are generally small, they yield to low network utilization rates. However, the combined effect of BNC CPU processing and network is less than its ODM counterpart, since BNC avoids expensive disk operations.
- PNC throughput values fall below ODM performance, while BWC and PWC configurations present the worst performance rates. PNC is a hybrid between ODM and BNC. In PNC, server logs are maintained and client originated queries retrieve log portions on-demand (similarly to ODM). In addition, at regular time intervals (every 5 secs.), a daemon for update broadcasting is invoked and propagates without discrimination the updated tuples "not sought" until that time. Since there is no think time, the disk utilization for this policy ranges between 0.91 and 0.94 for more than 30 clients. This forces the throughput curve to be considerably lower than that of ODM. The reason for the low throughput rates achieved by both BWC and PWC is the high CPU server utilization, which ranges between 0.73 and 0.78 for the BWC, and 0.53 and 0.62 for PWC for more than 25 clients. A great deal of the CPU time in BWC / PWC policies goes to processing of catalog pages. In particular, BWC spends 73.00 percent of its busy CPU time processing catalog pages and PWC 61.90 percent.

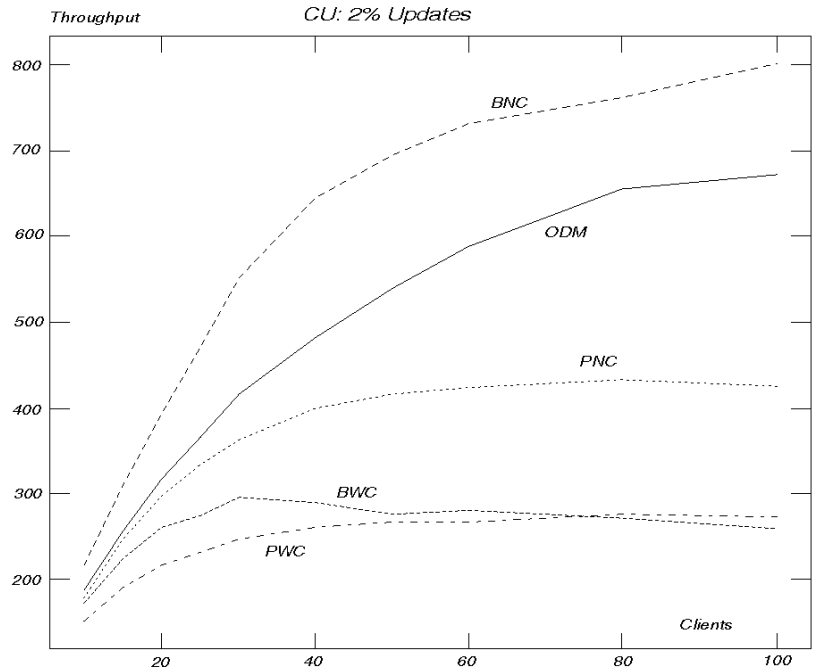

Fig. 9. CU experiment with 2 percent update jobs.

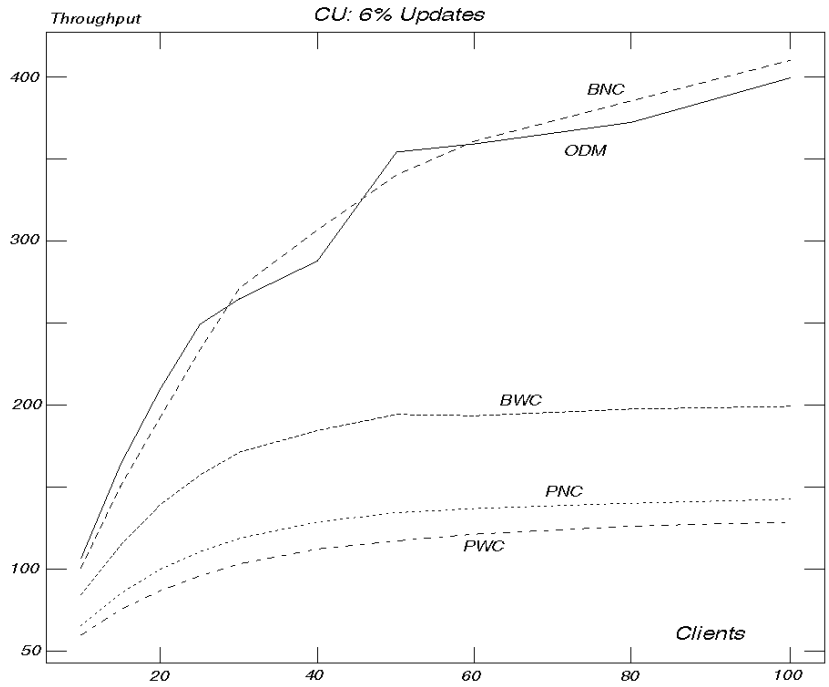

Fig. 10. CU Experiment with 6 percent update jobs.

- For more than 80 clients, PWC offers slightly improved throughput rates over BWC. In this client space, both configurations achieve similar server CPU and network utilizations. However, PWC retrieves portions of the logs at regular intervals and this results in high disk utilization levels (between 0.95 and 0.97), while the BWC's disk utilization remains limited (between 0.49 and 0.51). Higher disk utilization indicates that, while the CPU is processing, either updates or network related requests, the disk manager forwards the appropriate logs portions to be transmitted into the buffer area. This concurrent activity is the reason why PWC offers better throughput rates.

Fig. 10 shows the results of the CU experiment with writers updating 6 percent of the server relation pages. BNC and ODM curves come very close, since the benefits and penalties of each one under the current size of updates

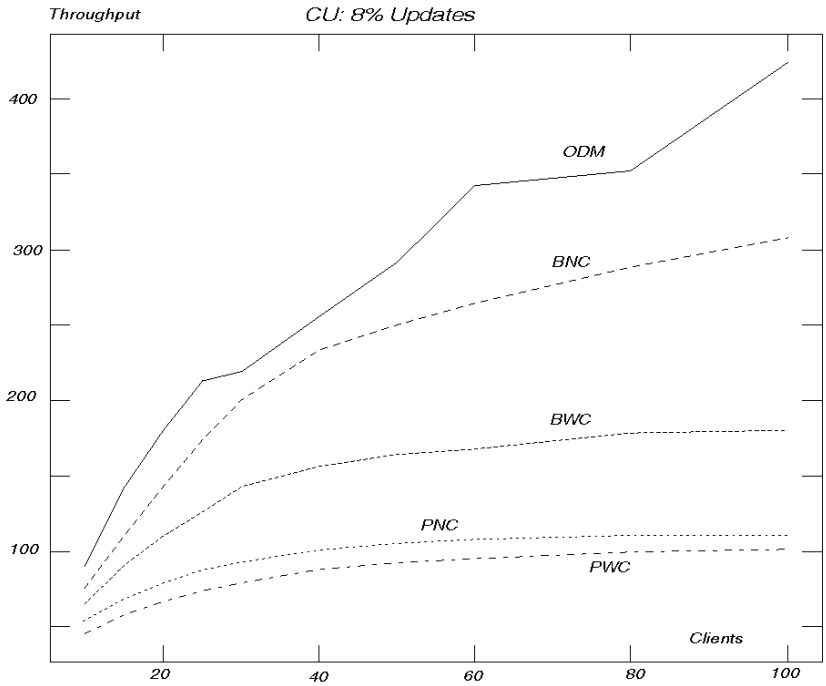

Fig. 11. CU experiment with 8 percent update jobs.

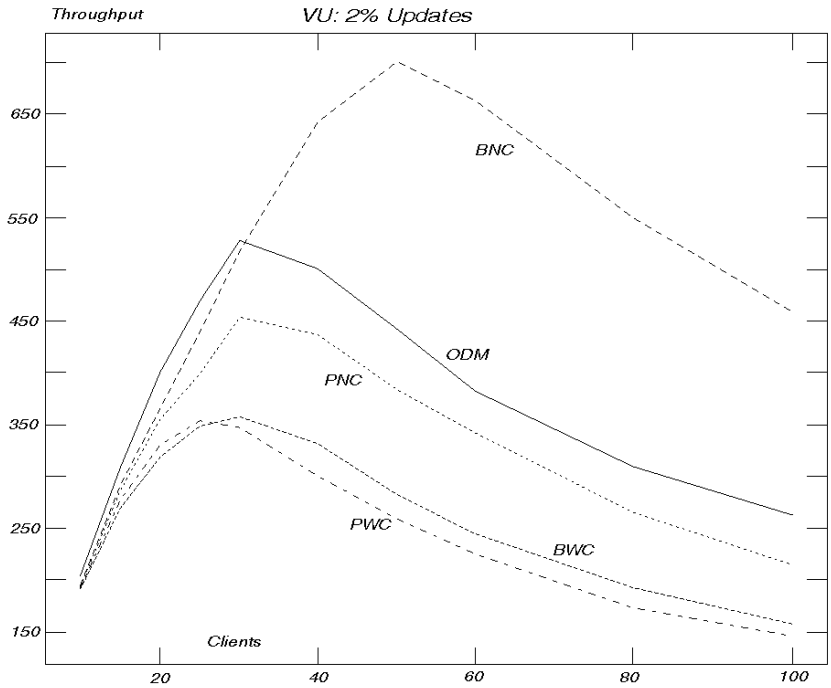

Fig. 12. VU experiment with 2 percent update jobs.

provide almost equivalent throughput rates (note that the total number of update jobs remains the same as before). Essentially, the higher network utilization along with the higher CPU server utilization offset the high disk utilization of the ODM (the latter still maintains significantly lower CPU and network utilization). PNC and PWC drop below the BWC curve as the sizes of the logs increase (due to larger updates) and create more disk accesses for both types of periodic propagation.

Fig. 11 depicts the results for the CU experiment with 8 percent update page selectivity jobs. The ODM curve gives better results than that of BNC. Due to the significantly larger number of updated tuples, BNC creates a congested network. The ODM configuration maintains low network utilization by selectively forwarding only portions of the logs. The gap between BNC and BWC becomes smaller compared to the corresponding gaps of the two previous graphs. The same is the case with PNC and PWC. This

TABLE 4
DISK, CPU, AND NETWORK UTILIZATION RATES FOR ODM AND BNC

$Num.\ of\ Clients \rightarrow$	10	20	30	40	50	60	80	100
$ODM\ \ Disk-Ut$	0.18	0.45	0.74	0.89	0.77	0.96	0.97	0.97
$ODM\ \ CPU-Ut$	0.07	0.19	0.34	0.44	0.40	0.50	0.51	0.48
$ODM\ \ Net-Ut$	0.07	0.14	0.19	0.20	0.15	0.17	0.14	0.12
$BNC\ \ Disk-Ut$	0.12	0.23	0.32	0.40	0.45	0.43	0.36	0.30
$BNC\ \ CPU-Ut$	0.04	0.11	0.22	0.34	0.46	0.51	0.55	0.56
$BNC\ \ Net-Ut$	0.09	0.24	0.43	0.64	0.83	0.91	0.96	0.98

indicates that heavy updates are handled better with directory-based techniques (BWC, PWC). Obviously, there is a trade-off between catalog page reading and update propagation over the network.

6.3 VU Experiment

Fig. 12 shows the results of the VU experiment for 2 percent updates in all five propagation configurations. The major points are:

- ODM dominates up to 30 clients, but then drops below the throughput rates achieved by BNC. Table 4 explains why this happens. ODM disk and CPU utilization values are overall higher than their counterparts of BNC, resulting in faster completion of the client jobs.
- ODM decline starts at 40 clients; beyond this point, the server disk utilization ranges between 0.87 and 0.97 (Table 4). This indicates that the disk becomes the main bottleneck point as the number of updates increases with the number of participating clients.
- BNC decline starts at 50 clients when the network utilization reaches 0.83. Beyond this point, the network utilization ranges between 0.91 and 0.98 and it becomes the major bottleneck element for the strategy.
- PNC achieves lower rates than ODM mainly due to stringent time conditions at the server, and the extra disk and CPU required processing for periodic update propagation.
- Policies based on catalog page reviewing have the worst performance. BWC requires heavy use of the server CPU for clusters that have more than 25 clients attached (CPU utilization is between 0.47 and 0.83), while PWC demonstrates a highly utilized disk manager (utilization is between 0.79 and 0.97 for more than 25 clients). The heavy PWC disk utilization in this area results in performance worse than that of BWC.

Fig. 13 shows the results of the experiment with updates of 6 percent. BNC offers the best performance throughout the range of the clients, while BWC has emerged as the second best configuration. These two configurations demonstrate high server disk and CPU utilizations while the ODM suffers from very high disk utilization for more than 30 clients (higher than 0.91). Periodic type of propagation policies suffer also from very high disk utilization rates for more than 15 clients. Similar trends are shown in Fig. 14 in which the various curves become more distinguished.

It is worth mentioning that, in the VU workload, where the total number of update jobs increases with the number of clients increases, the coupling of a fast server with a fast network[1] makes broadcasting a more effective way of pushing changes than the lazy and on-demand strategy. ODM has to spend a considerable amount of time in the disk resident log.

6.4 Experiments with Think Time

To examine the behavior of the various propagation policies in the presence of nonzero think time we reran the experiments with an average client think time of 15 secs. For brevity, we present only four resulting graphs, namely those corresponding to the experiments CU and VU for update jobs with page selectivity of 2 percent and 6 percent.

Figs. 15 and 16 depict the results of the CU experiment. ODM is does better than any other configuration since the think time provides lighter server resource contention. In Fig. 16, ODM offers inferior throughput rates than BNC in the 80-100 client range due to high disk utilization (which varies between 0.88 and 0.90). In this same high client space, the BNC strategy capitalizes on the fast network interface and provides better throughput rates. ODM maintains lower throughput rates only when the server disk becomes the bottleneck (Fig. 16—between 80 and 100 clients). The configurations based on the periodic type of update propagation perform well and their performance approaches that of ODM in Fig. 15. PNC and PWC use the server's idle periods (implicitly provided by the client think time) to push updates. However, under larger updates (i.e. Fig. 16), the gap between PNC/PWC and ODM becomes larger since these idle server periods become shorter. Note also that in Fig. 16, PNC/PWC give better results than BNC/BWC due to light server resource utilization in the 5-25 client range.

Figs. 17 and 18 show the results of the VU with 2 percent and 6 percent update jobs and an average client think time of 15 secs. In the graph with 2 percent update jobs, the ODM offers the best performance between five and 80 clients. Beyond 80 clients, its throughput rates are inferior to those of BNC due to heavy server resource utilization. Simple updated tuple broadcasting does relatively well at the beginning of the client space, and offers the best rates for more than 80 clients (the network for BNC is still fairly uncongested, i.e., at 100 clients the utilization is 0.34). In Fig. 17, the BWC configuration gives the poorest rates. The number of binding information pages that have to be retrieved in order to process updates increases linearly with the number of submitting streams (VU experiment). This retrieval activity contributes significantly to the deterioration of the throughput rates.

1. 10MBits/sec are allotted effectively to transfer modified tuples/data between server and clients.

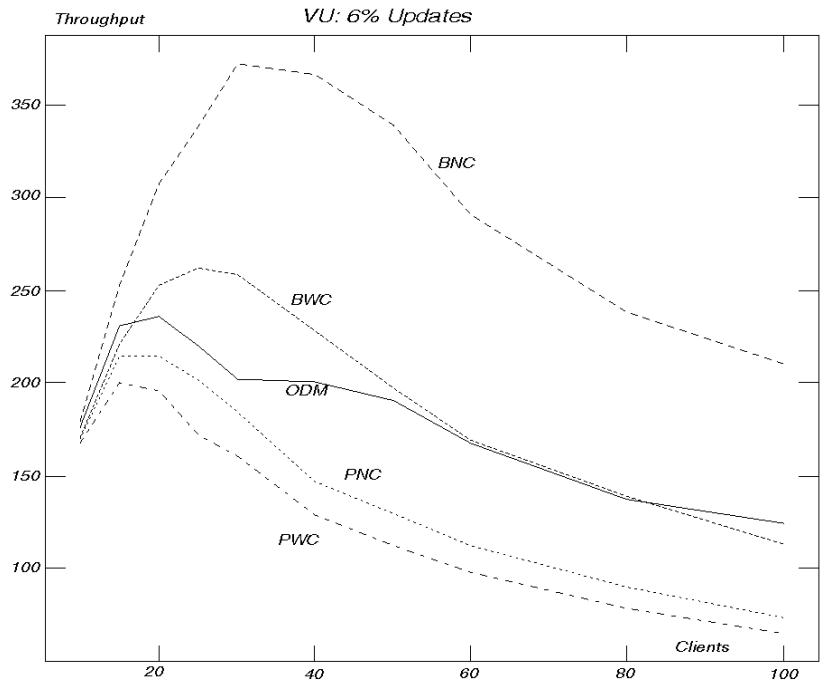

Fig. 13. VU experiment with 6 percent update jobs.

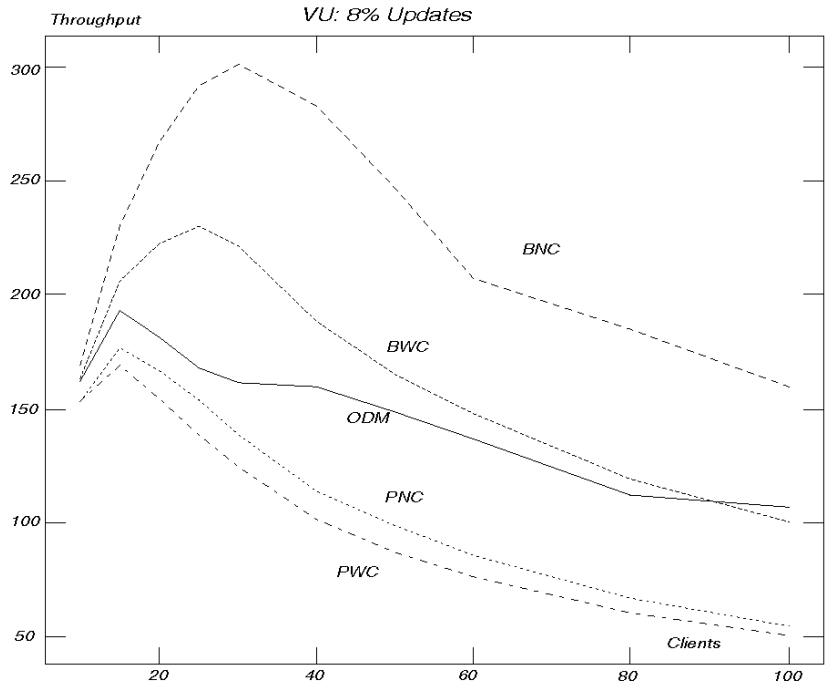

Fig. 14. VU experiment with 8 percent update jobs.

When the updates become larger (Fig. 18), the decline of ODM over BNC comes much earlier—at around 50 clients—since the server log manager must cope with larger pieces of updated data. The server disk utilization varies between 0.69 and 0.89. In the range between 60 and 90 clients, the BWC behaves better than ODM because its server resources remain moderately loaded (i.e., disk utilization varies between 0.49 and 0.56 and CPU between 0.59 and 0.76). Beyond that point, the BWC becomes CPU-bound and the ODM disk-bound. Both BWC and ODM provide similar throughput rates. PNC/PWC present the poorest performance as they not only use the log manager heavily but also utilize the server's CPU a great deal.

6.5 Sensitivity Analysis

In this last section, we examine the behavior of the five update propagation strategies under diverse network and catalog paging parameter settings. Fig. 19 depicts the results of the CU experiment, with job updates of 8 percent in a highly loaded network. We simulate this congested net-

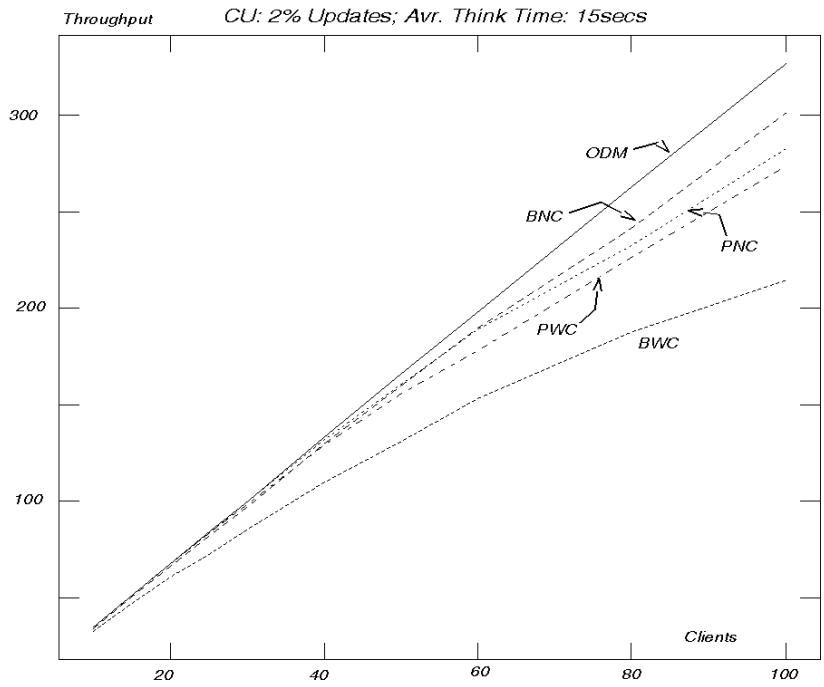

Fig. 15. CU experiment with 2 percent update jobs and think time.

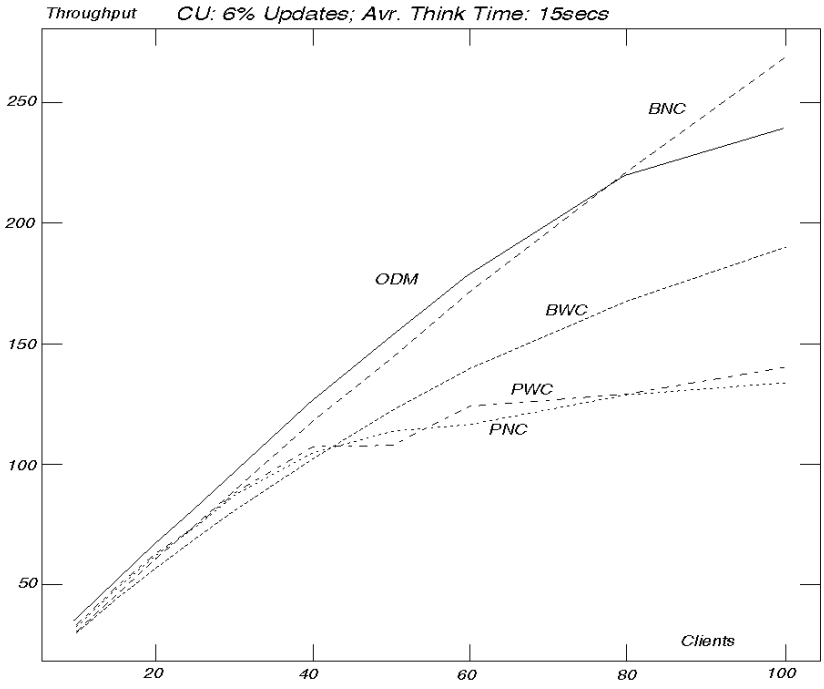

Fig. 16. CU experiment with 6 percent update jobs and think time.

work by setting the *msg_const* overhead to 50,000 instructions and bringing the effective transfer rate over the network at 0.5 Mbits/Sec. In this graph, the ODM configuration offers the best throughput rates, while the configurations based on filtering results after consulting catalog pages (BWC and PWC) come second. Overall, ODM maintains 1.8 times better average performance than PWC and 4.14 times better than BNC due to effective use of the incremental log operations. The network utilization in PNC/BNC for more than 10 clients is more than 0.97, and around the same levels in PWC/BWC for more than 40 clients. In contrast, the ODM configuration maintains network utilization between 0.13 (at 10 clients) and 0.70 (at 100 clients). For the other update curves (2 percent, 4 percent, and 6 percent), we observe similar trends with those depicted in Fig. 19.

Fig. 20 shows the results for the respective VU experiment with 8 percent update jobs in a highly loaded net-

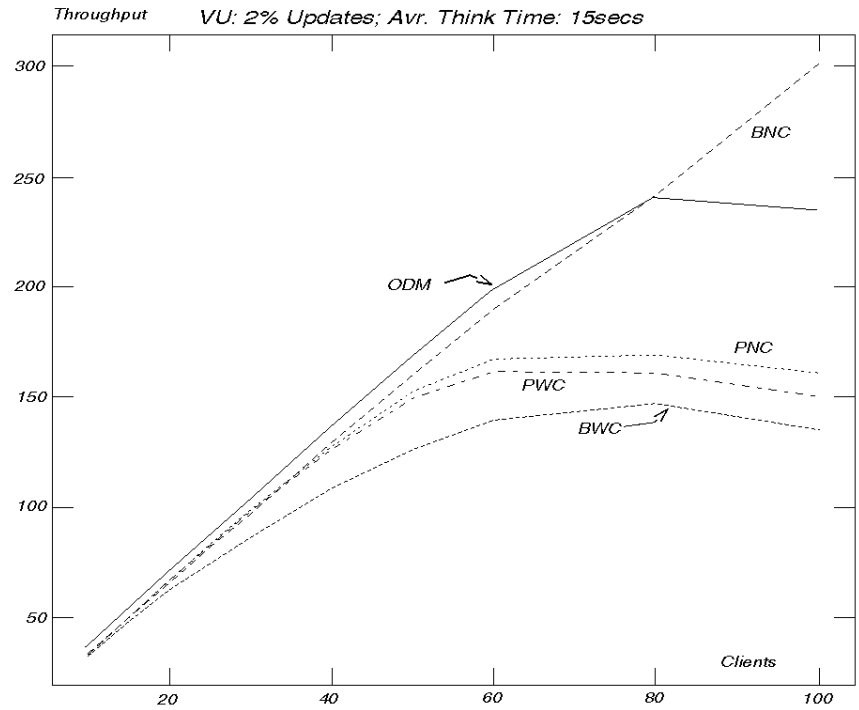

Fig. 17. VU experiment with 2 percent update jobs and think time.

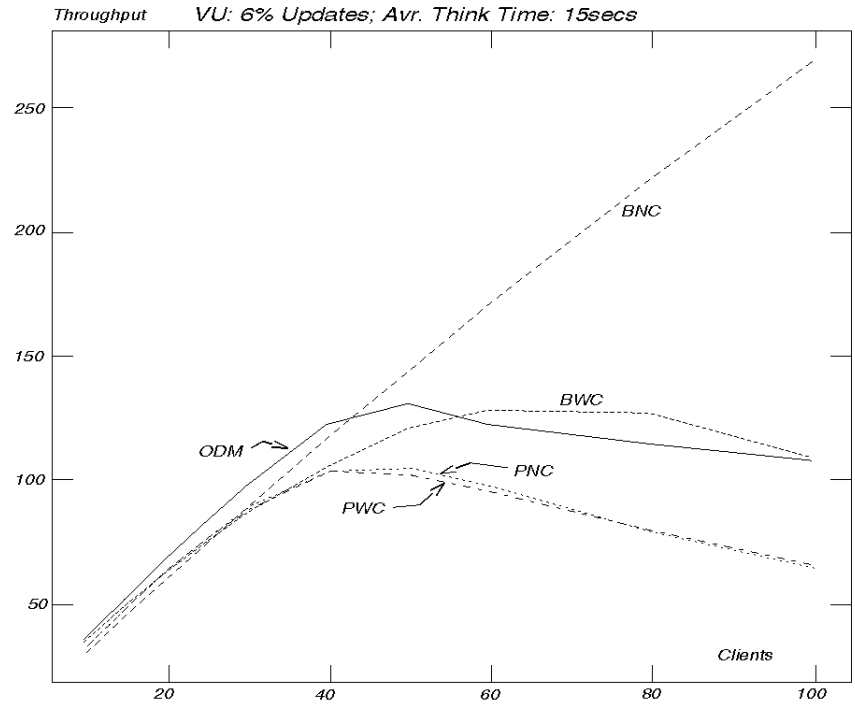

Fig. 18. VU experiment with 6 percent update jobs and think time.

work. BWC offers better rates than ODM in the range between 10 and 40 clients. In this range, the high disk utilization observed for ODM creates significant overheads. Nevertheless, for more that 40 clients, BWC experiences high network utilization (greater than 0.97). High network utilization works negatively for the BWC policy in the high client space since the total number of updates increases linearly to the number of participating clients (VU type of experiment). The ODM network utilization ranges between 0.26 and 0.64 in the whole client space of the experiment.

Figs. 21 and 22 show the results of CU and VU experiments for updates of 2 percent in a long haul or mobile network. Clients and server communicate through dedicated communication lines at 19,600 bits per second (BPS). ODM in both experiments offers the best service since it uses its incremental log processing. In both experiments, the network

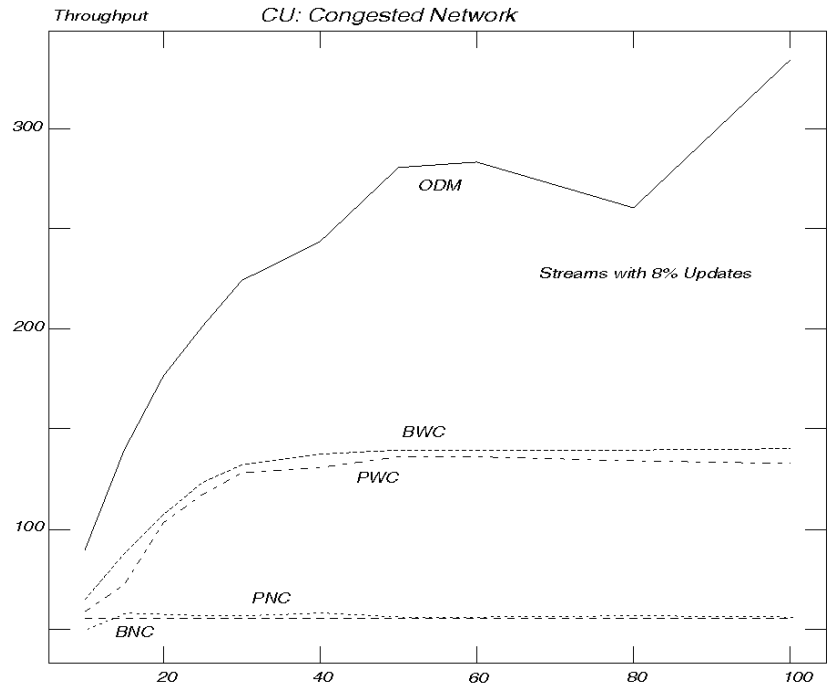

Fig. 19. CU experiment with highly loaded network (8 percent update jobs).

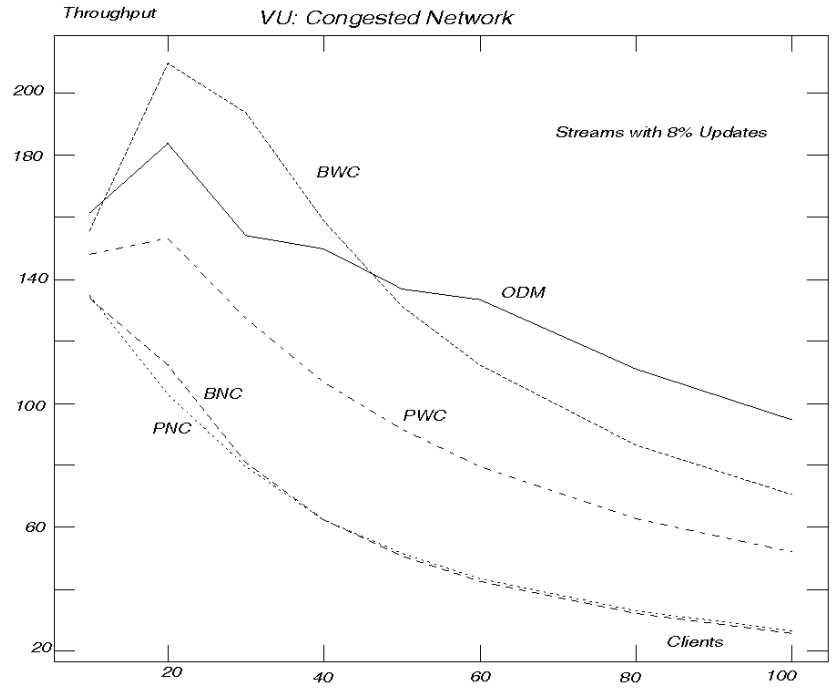

Fig. 20. VU experiment with highly loaded network (8 percent update jobs).

bandwidth is almost fully utilized in all configurations. Thus, the best strategy is the one that puts the least amount of traffic on the network (i.e., ODM). Although both PWC and BWC discriminate in terms of the volume of data they put forward to the network, they fail to service clients individually. This creates longer completion times for the submitted streams than those achieved in ODM.

Finally, Figs. 23 and 24 present the results of the CU and VU experiments under light penalties for catalog-based operations for broadcasting policies. More specifically, the server makes one disk access to retrieve the binding conditions for a group of five clients on average, and each such page is processed with 0.125 msec *cpu_dir_page* overhead once in the buffer area. Although the curves in Figs. 23 and 24 indicate trends similar to those of Figs. 19 and 20, the BWC and PWC have come very close to the ODM which maintains the best overall performance. This observation indicates that catalog-based propagation techniques are worth exploiting as long as the catalog bookkeeping is inexpensive. In CU and for clients in the range of 10-20, BWC

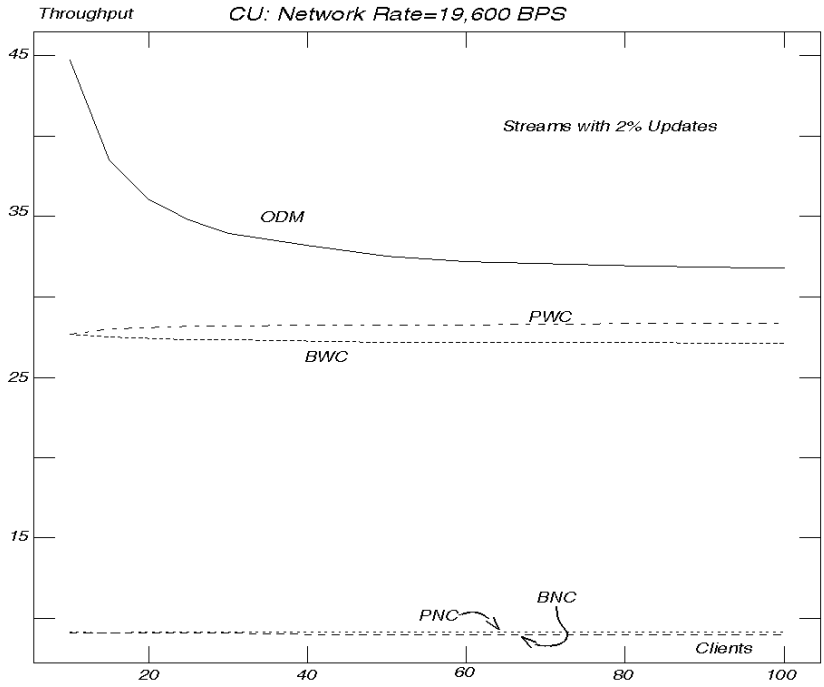

Fig. 21. CU experiment with network rate 19,600 BPS.

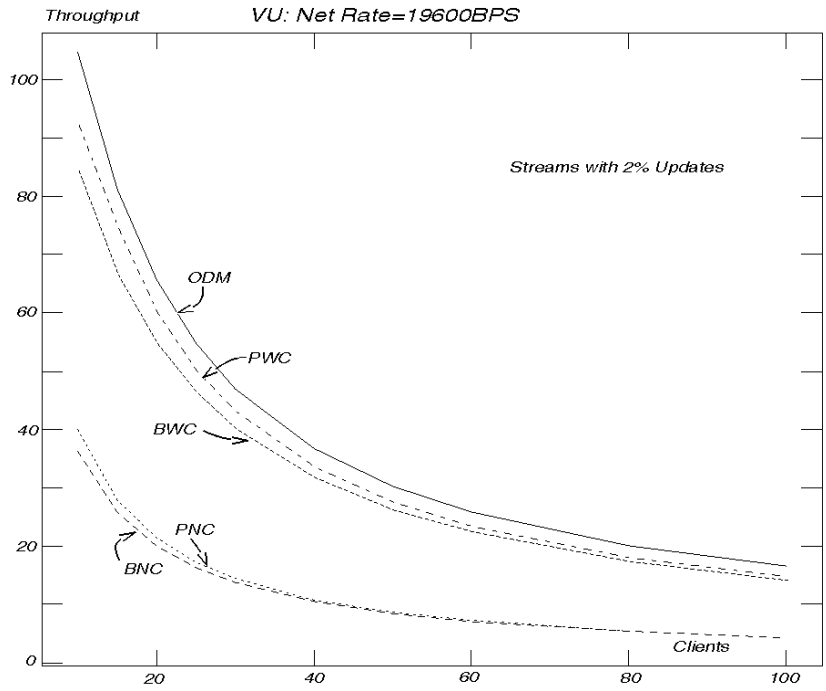

Fig. 22. VU experiment with network rate 19,600 BPS.

gives higher throughput values than its ODM counterpart because of the available network bandwidth. High network utilization is a serious obstacle for achieving higher performance rates in both BWC and PWC for more than 40 clients.

7 Conclusions

Contemporary Client-Server DBMS architectures do not only exploit ephemeral data caching but they also make use of the available client disk space. The Enhanced Client-Server DBMS is such a configuration where clients cache data from the server(s) in their long-term memory. However, data consistency needs to be maintained at all times. In this paper, we have discussed update propagation techniques for the Enhanced Client-Server DBMS architecture and evaluated them under multiple job streams of different composition and varying update rates.

Five strategies for propagating updates from the server to the clients were proposed, namely, ODM (On Demand),

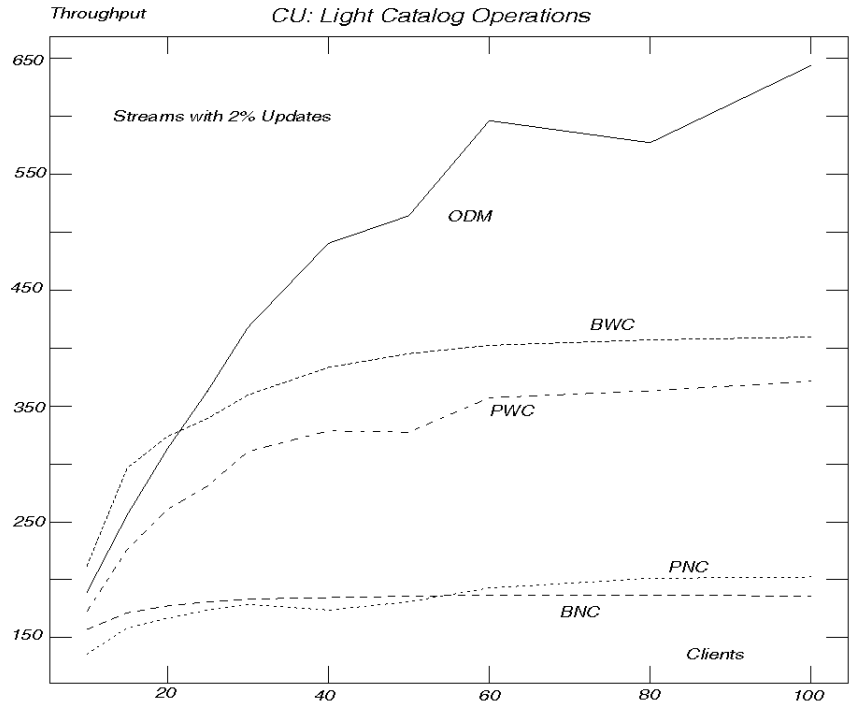

Fig. 23. CU experiment with inexpensive catalog access operations.

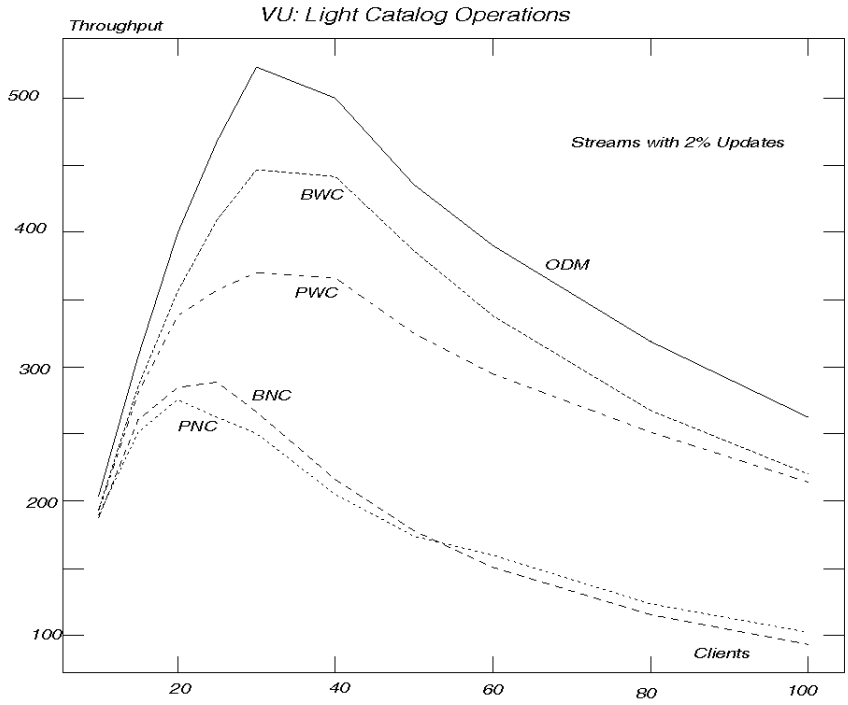

Fig. 24. VU experiment with inexpensive catalog access operations.

BNC/BWC (Broadcasting with NO/With Catalog bindings), and PNC/PWC (Periodic broadcasting with NO/With Catalog bindings). The core architectural configuration for our experiments consisted of a server connected to a varying number of clients. We were interested in the way that the various update propagation strategies scale up their performance as the number of clients per server increases. Our main experimental results are:

1) ODM offers the best performance if none of the server resources reaches full utilization.
2) Under high utilization of server resources, the BNC configuration surprisingly offers the best performance when:
   - The updates have small update page selectivities.
   - The number of clients is large (more than 60-70) in the CU family of experiments.

- The number of updates increases linearly with the number of clients attached to the server.

A fast local area network paired with fast processing CPUs at both ends of a critical path offers a combined job completion time for the broadcasting policies that is shorter than that achieved by the ODM strategy.

3) If ECS operates under a heavily loaded network, then ODM policy provides the best performance independent of workload. The gains become more obvious for the more heavily updating curves. The same is the case if ECS functions in long haul networks.
4) When server bookkeeping is inexpensive in terms of disk accesses and CPU processing time, propagation techniques based on catalog pages and updated tuple filtering may considerably cut down on network traffic.
5) Periodic type of update propagation demonstrates significant gains when there is nonzero think time. The highest gains were attained for the light update curves.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their comments that helped us significantly improve the presentation of the paper and Steve Milliner for many discussions. A preliminary version of this paper was presented in [16]. This work was supported in part by the Center for Advanced Technology in Telecommunications (CATT) in Brooklyn, New York, and grants from NASA NAGW-2777, U.S. National Science Foundation EEC 94-02384, IIS-9733642, and ARPA F30602-93-C-0177.

REFERENCES

[1] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data Caching Issues in an Information Retrieval System," *ACM Trans. Database Systems*, vol. 15, no. 3, pp. 359-384, Sept. 1990.
[2] J. Archibald and J.L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. Computer Systems*, vol. 4, no. 4, Nov. 1986.
[3] *Building An Object-Oriented Database System: The Story of O*$_2$, F. Bancilhon, C. Delobel, and P. Kanelakis, eds. San Mateo, Calif.: Morgan Kaufmann, 1992.
[4] S. Banerjee and P.K. Chrysanthis, "Data Sharing and Recovery in Gigabit-Networked Databases," *Proc. Fourth Int'l Conf. Computer Comm. and Networks*, Las Vegas, Nev., Sept. 1995.
[5] M. Bellew, M. Hsu, and V. Tam, "Update Propagation in Distributed Memory Hierarchy," *Proc. Sixth Int'l Conf. Data Eng.*, pp. 521-528, Los Angeles, 1990.
[6] A. Bhide and M. Stonebraker, "An Analysis of Three Transactions Processing Architectures," *Proc. 14th Very Large Data Base Conf.*, pp. 339-350, Los Angeles, 1988.
[7] J. Bloomer, *Power Programming with RPC*. Sebastopol, Calif.: O'Reilly and Associates, 1992.
[8] *Database Machines*, H. Boral and P. Faudemay, eds. Springer-Verlag, June 1989.
[9] P. Butterworth, A. Otis, and J. Stein, "The Gemstone Object Database Mangement System," *Comm. ACM*, vol. 34, no. 10, Oct. 1991.
[10] M. Carey, M. Franklin, M. Livny, and E. Shekita, "Data Caching Tradeoffs in Client-Server DBMS Architecture," *Proc. ACM-SIGMOD Conf. Management of Data*, Denver, Colo., May 1991.
[11] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*. New York: McGraw-Hill, 1984.
[12] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145-186, June 1994.
[13] D.D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Comm.*, June 1989.
[14] A. Delis and N. Roussopoulos, "Performance and Scalability of Client-Server Database Architectures," *Proc. 19th Int'l Conf. Very Large Databases*, Vancouver, B.C., Canada, Aug. 1992.
[15] A. Delis and N. Roussopoulos, "Performance Comparison of Three Modern DBMS Architectures," *IEEE Trans. Software Eng.*, vol. 19, no. 2, pp. 120-138, Feb. 1993.
[16] A. Delis and N. Roussopoulos, "Management of Updates in the Enhanced Client-Server DBMS," *Proc. 14th IEEE Int'l Conf. Distributed Computing Systems*, Poznan, Poland, June 1994.
[17] U. Deppisch and V. Obermeit, "Tight Database Cooperation in a Server-Workstation Environment," *Proc. Seventh IEEE Int'l Conf. Distributed Computing Systems*, pp. 416-423, June 1987.
[18] D. DeWitt et al., "GAMMA—A High Performance Backend Database Machine," *Proc. 12th Conf. Very Large Data Bases*, Kyoto, Japan, Aug. 1986.
[19] D. DeWitt, D. Maier, P. Futtersack, and F. Velez, "A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems," *Proc. 16th Very Large Data Bases Conf.*, pp. 107-121, Brisbane, Australia, 1990.
[20] D. Ferrari, *Computer Systems Performance Evaluation*. Englewood Cliffs, N.J.: Prentice Hall, 1978.
[21] M. Franklin, M. Carey, and M. Livny, "Local Disk Caching in Client-Server Database Systems," *Proc. 19th Int'l Conf. Very Large Data Bases*, Dublin, Ireland, Aug. 1993.
[22] M. Franklin, M. Zwilling, C. Tan, M. Carey, and D. DeWitt, "Crash Recovery in Client-Server EXODUS," *Proc. ACM -SIGMOD Conf.*, San Diego, Calif., June 1992.
[23] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. San Mateo, Calif.: Morgan-Kaufman, 1992.
[24] R. Hagman and D. Ferrari, "Performance Analysis of Several Back-End Database Architectures," *ACM Trans. Database Systems*, vol. 11, no. 1, pp. 1-26, Mar. 1986.
[25] B. Kim and P. Wang, "ATM Networks: Goals and Challenges," *Comm. ACM*, vol. 38, no. 2, Feb. 1995.
[26] W. Kim, J. Garza, N. Ballou, and D. Woelk, "Architecture of the Orion Next-Generation Database System," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, no. 1, pp. 109-124, Mar. 1990.
[27] S. Kleiman, D. Shah, and B. Smaalders, *Programing with Threads*. Mountain View, Calif.: SunSoft Press / Prentice Hall PTR, 1995.
[28] K. Küspert, P. Dadam, and J. Gunauer, "Cooperative Object Buffer Management in the Advanced Information Management Prototype," *Proc. 13th Very Large Data Bases Conf.*, Brighton, U.K., 1987.
[29] C. Lamb, G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore Database System," *Comm. ACM*, vol. 34, no. 10, Oct. 1991.
[30] D. Marakoff and D. Eager, "Disk Cache Performance for Distributed Systems," *Proc. 10th IEEE Int'l Conf. Distributed Computing Systems*, pp. 212-219, Paris, May 1990.
[31] D. McGovern and C.J. Date, *A Guide to SYBASE and SQL Server*. Reading, Mass.: Addison-Wesley, 1992.
[32] S. Milliner and A. Delis, "Networking Abstractions and Protocols Under Variable Length Messages," *Proc. 1995 IEEE Int'l Conf. Network Protocols (ICNP-95)*, Tokyo, Nov. 1995.
[33] C. Mohan and I. Narang, "ARIES / CSA: A Method for Database Recovery in Client-Server Architecture," *Proc. Conf. Management of Data SIGMOD*, Minneapolis, Minn., June 1994.
[34] A. Nakamura and M. Takizawa, "Reliable Broadcast Protocol for Selectively Ordering PDUs," *Proc. 11th IEEE Int'l Conf. Distributed Computing Systems*, Arlington, Tex., June 1991.
[35] T. Ng, "Propagating Updates in a Highly Replicated Database," *Proc. Sixth Int'l Conf. Data Eng.*, pp. 529-536, Los Angeles, 1990.
[36] B. Nitzberg and V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms," *Computer*, vol. 24, no. 8, pp. 522-60, Aug. 1991.
[37] S. Ough and R. Sonnier, "Spotlight on FDDI," *Unix Review*, vol. 10, no. 10, pp. 40-49, Oct. 1992.
[38] E. Panagos, A. Biliris, H.V. Jagadish, and R. Rastogi, "Client-Based Logging for High Performance Distributed Architectures," *Proc. 12th Int'l Conf. Data Eng.*, pp. 344-351, New Orleans, Feb.-Mar. 1996.
[39] N. Roussopoulos, "The Incremental Access Method of View Cache: Concept, Algorithms, and Cost Analysis," *ACM Trans. Database Systems*, vol. 16, no. 3, pp. 535-563, Sept. 1991.

[40] N. Roussopoulos, C. Chen, S. Kelley, A. Delis, and Y. Papakonstantinou, "The ADMS Project: Views R Us," *IEEE-Bulletin Data Eng.*, vol. 18, no. 2, pp. 19-28, June 1995.
[41] N. Roussopoulos and H. Kang, "Principles and Techniques in the Design of *ADMS±*," *Computer*, vol. 19, no. 12, pp. 19-25, Dec. 1986.
[42] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Trans. Computers*, vol. 39, no. 4, Apr. 1990.
[43] M. Schroeder and M. Burrows, "Performance of Firefly RPC," *ACM Trans. Computer Systems*, vol. 8, no. 1, pp. 1-17, Feb. 1990.
[44] A. Segev and J. Park, "Maintaining Materialized Views in Distributed Databases," *Proc. Fifth Int'l Conf. Data Eng.*, pp. 262-270, Los Angeles, 1989.
[45] L. Shapiro, "Join Processing in Database Systems with Large Main Memories," *ACM Trans. Database Systems*, vol. 11, no. 3, Sept. 1986.
[46] A. Sinha, "Client-Server Computing," *Comm. ACM*, vol. 35, no. 7, July 1992.
[47] A. Stamenas, "High Performance Incremental Relational Databases," master's thesis, Dept. of Computer Science, Univ. of Maryland, College Park, 1989.
[48] R. Stevens, *Unix Networking Programming*. Englewood Cliffs, N.J.: Prentice Hall, 1990.
[49] M. Stonebraker, "Object-Relational DBMS—The Next Wave," technical report, Menlo Park, Calif., 1996. http://www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm.
[50] L. Svobodova, "File Servers for Network-Based Distributed Systems," *Computing Surveys*, vol. 16, no. 4, pp. 353-398, Dec. 1984.
[51] R. Velter, C. Spell, and C. Ward, "Mosaic and the World-Wide Web," *Computer*, vol. 27, no. 10, Oct. 1994.
[52] Y. Wang and L. Rowe, "Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture," *Proc. 1991 ACM SIGMOD Int'l Conf.*, Denver, Colo., May 1991.
[53] K. Wilkinson and M.A. Neimat, "Maintaining Consistency of Client-Cached Data," *Proc. Int'l Conf. Very Large Data Bases*, Brisbane, Australia, Aug. 1990.

Alex Delis received his PhD and his MSc in computer science from the University of Maryland at College Park and his BS in computer engineering from the University of Patras, Greece. Dr. Delis is currently an assistant professor in the Department of Computer and Information Science at Polytechnic University in Brooklyn, New York. His research interests are in databases, computer systems, and software engineering. He is a member of the IEEE, the ACM, Sigma Xi, the New York Academy of Sciences, and has received the U.S. National Science Foundation's Career Award.

Nick Roussopoulos is a professor in the Computer Science Department and at the Institute for Advanced Computer Studies at the University of Maryland, College Park. He served on the Space Science Board Committee on Data Management and Computation (CODMAC) from 1985 until 1988. He was the general chair of the ACM International Conference on Data Management in 1986. He was an elected trustee of the VLDB Endowment from 1990-1996. He is a member of the editorial board of the *International Journals on Information Systems*, *Decision Support Systems*, and *Intelligent Cooperative Information Systems* (*IJICIS*). Dr. Roussopoulos is the principal investigator of the ADMS+- project at the University of Maryland and has published more than 80 refereed papers in journals and conference proceedings. His research area is in database systems, data warehousing, client-server database architectures, heterogeneous databases and interoperability, mobile databases, data broadcast, network management systems, and geographic information systems.