Management of Updates in the Enhanced Client–Server DBMS *

Alex Delis
School of Information Systems
Queensland Univ. of Technology
Brisbane, QLD 4001, Australia

Nick Roussopoulos[†]
Computer Science Department
University of Maryland
College Park, MD 20742, USA

Abstract

The Client–Server DBMS model has emerged as the main paradigm in database computing. The Enhanced Client–Server architecture takes advantage of all the available client resources including their disk managers. Clients can cache server data into their own disk units if data are part of their operational space. However, when updates occur at the server, some of the client data managers may need to not only be notified about them but also obtain portions of the updates as well. In this paper, we examine the problem of managing server imposed updates that affect client cached data. 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.

1 Introduction

In recent years, we have seen a number of important technology developments, namely, the wide availability of inexpensive workstations and PCs, the introduction of large, fast and reliable disk units, as well as the appearance of fast local area networks (FDDI networks [19]). These developments paved the way to the introduction of the Client–Server Database Systems (CS–DBMSs).

The central concept in CS-DBMSs is that a dedicated machine runs a DBMS and maintains a main centralized database (DBMS–Server)[10, 8]. The users of the system access the database through either their workstations or PCs via a local area network. They usually run their programs locally on their own machines and direct all database inquiries and/or updates to the DBMS–Server. In this way, they become the server's clients. This configuration is termed Standard Client–Server DBMS (SCS) [7]. Although the environment in SCS is distributed, the DBMS is centralized and therefore, transaction handling is easier than distributed transaction management. The computational Client–Server model [24] has been adopted as a standard by DBMS suppliers and used in commercial products [15, 13] as well as a number of research prototypes [12, 3]. The Enhanced Client–Server DBMS [8] off–loads disk accesses from the server by having the clients run a limited DBMS, in terms of concurrency, and by caching results of server queries to the client disk managers. In this paper, we study propagation strategies of server imposed updates for the Enhanced Client–Server DBMS architecture as the number of participating clients increases. A number of update propagation techniques is presented, and their performance is studied through simulation under various workloads.

The paper is organized as follows: the second section gives an overview of the Enhanced Client–Server DBMS. In section 3, we state the problem and compare this work with prior related studies. The fourth section proposes a number of policies, and discusses unique policy characteristics and overheads. Section 5 gives an overview of the simulation models used. In section 6, we present the evaluation methodology and some results from our simulation experiments. Finally, conclusions can be found in the last section.

2 Overview of the Enhanced Client–Server DBMS Architecture

The Standard Client–Server DBMS architecture (SCS) uses the network as the means to either send messages or ship query results from the server to clients. Heavy database processing can create serious overheads on the server. The Enhanced Client–Server DBMS architecture (ECS) utilizes both the CPU and the I/O of the client by caching query results and by enhancing the client functionality to a DBMS for incremental management of cached data [7]. Figure 1 depicts this setting. Although the architecture can be generalized for any data model we restrict our discussion to the relational model where we have already defined incremental operations [20].

Caching query results over time permits a client to cre-

*This research was partially supported by grants NRGS94-18-16-500-06, NASA/USRA FCPO 550-81, NSF IRI-8719458, and GDR-85-00108.

†Also with the University of Maryland Institute for Advanced Computer Studies (UMIACS).

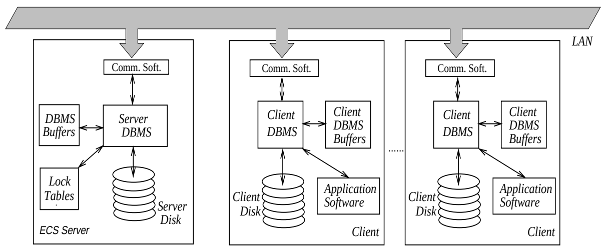

Figure 1: The ECS DBMS Architecture

ate a server database subset on its own disk unit. A client database is a partial replica of the server database that is of interest to the client's application(s). There are two major advantages for this kind of disk caching: firstly, requests for the same data from the server are eliminated and secondly, system performance is boosted since the CPU of clients can access local data copies. Nonetheless, in the presence of updates the system needs to ensure proper propagation of new item values to the appropriate clients.

When the result of a query is cached into a local relation for the first time, this "new" local relation is *bound* to the server relations used in extracting the result. Every such binding is described by three elements: the participating server relation(s), the applicable condition(s) on the relation(s), and a timestamp. The condition is essentially the filtering mechanism that decides what are the qualifying tuples for a particular client. The timestamp indicates the last time that a client has seen the server updates that may affect its cached data. Bindings can be either stored at the server's catalog or maintained by the individual clients.

Updates are 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. Every server relation is associated with an update propagation log which consists of timestamped inserted tuples and timestamped qualifying conditions 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 [23].

Client query processing against bound data is preceded by a request for an incremental update of the cached data. The server is required to look up log portions of the query involved relations. These log portions maintain timestamps greater than the one currently held by the client. The look up process may be done once the binding information for the demanding client is available. Only relevant fractions or increments of the modifications (relation update logs) are propagated to the client's site. The set of algorithms that carry out these tasks are based on the incremental access methods for relational operators described in [20].

The concurrent processing of all updates and query/log operations is carried out by the Server DBMS. In order to maintain consistency, data pages are accessed through a standard locking protocol such as the 2ϕ locking protocol. We assume that DBMS buffer areas used in both server(s) and clients can accommodate only portions of their disk–resident data at a time. By making the server the primary site of the configuration, we avoid all the complex job management and processing that has to be carried out by a conventional distributed DBMS.

3 The Problem and Related Work

The main question addressed in this paper could be summarized as follows: given an ECS–DBMS configuration and a server committed update, what are the best alternatives in propagating the results of this operation to the interested clients? 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 relaxation (quasi) update propagation methods in information systems. Carey et al. [6], and Wang and Rowe [26] examine the performance of data consistency algorithms for maintaining consistency of data cached in clients' buffers. Franklin et al. examine the performance of various page–oriented caching techniques [11]. There is also a large amount of work done in the areas of cache coherence algorithms [2] and distributed shared main memories[18], where the major bottleneck point is the common bus.

When the problem is examined in the context of the ECS DBMS configuration, there is 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 at both server and clients. Few other questions that can be examined in this setting are: • What is the performance of the various propagation alternatives that we may employ ? • How do these strategies scale up in the presence of many clients (more than 30–40)? • Is there any gain in employing an incremental propagation strategy? • In an rare–update environment, there is no data inconsistency and clients work off their copies providing a system with almost linearly scalable performance. As updates increase and their operational areas on the database become larger, what is the overhead that needs to be paid by both clients and server to offer timely change propagation? Some work indirectly related to this study can be found in [17, 4, 22, 25, 14].

4 Description of the Strategies

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

On–Demand strategy (ODM): This policy has been essentially used in the model of the Enhanced Client–Server model outlined in section 2. The main idea is that the server does not do any book–keeping in terms of the data bindings. This implies that any time the client wants to answer a query it has to poll the server with its binding/caching information. In this way, the server is capable of identifying the data space of interest for every individual client and initiate the appropriate actions to service the request. Query messages, binding information, as well as update requests are directed through the network from the clients to the server. Data increments and update commit acknowledgments are forwarded from the server to the clients.

The second alternative strategy is built around the idea of broadcasting server data modifications to all clients in the cluster as soon as the update commits [16]. The rationale is that if the updated tuples are already in main memory, then we could avoid re–reading data from the disk when the need for update propagation arises. Thus, logs become unnecessary items. There are two alternatives for broadcasting data modifications: *Broadcasting with No Catalog bindings*, and *Broadcasting With Catalog bindings*.

Broadcasting with No Catalog bindings (BNC): This is the simple version of broadcasting in which committed updates are sent to all clients indiscriminately. This strategy requires no extra server functional overhead. As soon as a write operation commits, the server establishes a communication channel with each of its clients (one at a time). Through this channel, updated tuples (or pages) are shipped to workstations. When the client receives the changes, it suspends any on–going work and determines if the broadcasted modifications affect its operational locale in any way (this can be determined easily with the binding information at hand). If this is the case, the client aborts the current job (if any), flushes the newly arrived changes into its disk, and restarts the just aborted query. 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): The approach taken in this strategy is to limit the amount of broadcasted data over the network. This is done by reducing the volume of data based on server maintained binding information. A directory (or catalog) of binding information for each client has to be maintained in the server DBMS system area. This directory is a table of bindings that designate the specific areas of the database which 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 the client's binding calls for it. In addition, only a portion of the updated tuples needs to travel over the network, e.g. the one pertinent to the client. Any query executing at the time of broadcasting at the client site is aborted and can be restarted after the incoming modifications have been committed into the client disk manager. The directory of the bindings can be maintained in main memory. However, when the number of clients increases such an assumption may not be realistic and the binding directory has to be maintained on the disk.

Finally, there are two more possible propagation strategies 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" and client originated queries are handled in a manner similar to that described in the ODM strategy. The additional feature is that, at regular intervals, the server is interrupted by a daemon. This daemon essentially collects all the changes "not seen" by the clients so far, and initiates their propagation. This is done with the rationale that the strategy could gain on server idle time periods. During these periods some useful propagation work may take place. However, it is expected that under stringent job submission times (short think times) the periodic propagation suffers in comparison with the previous strategies. As soon as the server daemon reads the "not seen" portions of the log(s) into the memory buffers, it can dispatch them to the various clients. This can happen by either using a naive or a discriminatory broadcasting strategy. The former results into the *Periodic broadcasting with No Catalog bindings* (PNC) and the latter into the *Periodic broadcasting With Catalog bindings* (PWC). The qualitative difference between PNC and PWC is the same as that between BNC and BWC. PWC tries to limit the volume of data traveling over the local area network by using server catalog information about the operational areas of every client. Client queries in progress may be aborted and restarted after the modifications are applied on the client data.

5 Simulation Models

We have developed five software packages based on closed network queueing models corresponding to the five update propagation policies.

DBMS Operational Aspects	*Value*
page_size	2KBytes
srv_cpu_mips	41 MIPS
srv_disk_acc_tm	12 msec
srv_main_mem	2000 Pages
read_page	6500 ins
write_page	8000 ins
inst_sel	10000 ins
inst_prj	11000 ins
inst_join	29000 ins
inst_mod	12500 ins
inst_log	5000 ins
inst_ism	6000 ins
mpl	12
bl_delay	0.2 msec
dd_search	10 msec
kill_time	200 msec
cl_cpu_mips	20 MIPS
cl_disk_acc_tm	16 msec
cl_main_memory	500 Pages
Network Features	*Value*
rpc_{del}	10 msec
mesg_length	400 bytes
net_rate	10 Mbits/sec
Database Features	*Value*
Relation Size	1000 pages
Page_Size	2048 Bytes
Number of Relations	8
Caching – Logging Characteristics	*Value*
α_{Rel_i}	0.3
Write_Log_Fract	0.10
ISM_Proc	6000 ins

Table 1: Model Parameters

$AdditionalParameters$	$Meaning$	$Value$
net_proc	Avg. overhead for processing a page at NPM	2 msec
dir_cond	Avg. number of pages accessed for every directory search	2 pages
$cpu\_per\_dir\_page$	Average time needed to CPU a directory page	15 msecs
ODM_msg_len	Average Message Length for ODM–ECS Configuration	2 KBytes
$BRUP\_prc$	Avg. time to process a page of broadcasted updates	2.5 msec
msg_len	Average Message Length for non-ODM–ECS Configurations	400 Bytes
$per\_interval$	Periodic Broadcasting Interval	5 sec

Table 2: Additional Model Parameters

The original ECS closed network simulation model [8] is altered to reflect the changes that the above propagation strategies impose. There are three additional issues that are addressed by these queueing models, in particular: network processing, time spent for accessing and processing binding directory information, and filtering of broadcasted updates at the client sites wherever necessary. The description of the used closed network models is omitted for brevity here but it can be found in [9]. The set of parameters that describe the model elements are provided in Table 1. The α_{Rel_i} factor indicates the fraction of the server relation i that is cached in every client disk manager. Both clients and servers spent time for processing the critical path of RPCs [21, 5] in their *Network Proc. Modules*(NPM). Table 2 shows some additional parameters used in the models of this section. The *net_proc* parameter accounts for the extra CPU penalties that take place at the NPM processing elements.

The five simulation packages were written in C and their size vary between 5.3k to 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 47 hours of CPU time on a SparcStation 2.

6 Workloads and Experiments

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 have, we run two families of experiments: •Those with Constant number of Update jobs (CU), where a constant number of four streams submit updates and the remaining clients queries only (simulating stock market environments or generally environments with few writers and many readers). •Those with Variable Update jobs (VU) where each stream is a combination of both queries and updates—updates constitute 10% of all the jobs and are uniformly distributed over the queries (simulating traditional database environments). Queries

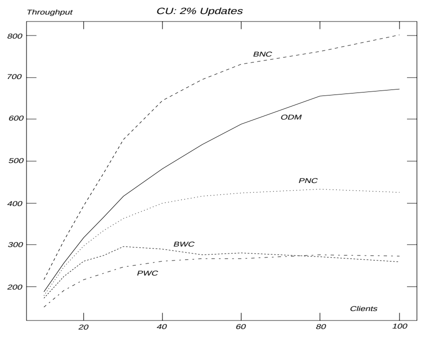

Figure 2: CU Experiment with 2% Update Jobs

consists of relational operations that manipulate up to 10% of the pages of the server relation(s). The same exactly streams were submitted for all update propagation strategies.

The objective of the experiments is twofold: first, to examine how the various update propagation techniques behave under these workloads, and second to identify important parameters and study their impact on the different strategies. In the experiments, we vary two parameters: the number of participating clients from 5 to 100 and the update page selectivity from 2% to 8%. 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 number of participating jobs per stream was selected to be long enough (135) to guarantee throughput confident results. The main performance criterion for our evaluation is the overall average job throughput. The average throughput is measured in jobs per minute (JPM). 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.

6.1 CU Experiment

Figure 2 shows the results of the five configurations for 2% update jobs in the CU workload. The number of update streams remains 4 throughout the experiment. BNC surprisingly performs better than ODM. In ODM strategy, there are log pages that need to be first flushed into the disk and then read on behalf of the various clients. In BNC, no such reading/writing takes place. Updated tuples from the main memory buffers (just before or after the commit) are transferred to system designated areas and put forward to the network interface. BNC charges the server CPU with some processing time and since the broadcasting happens in a point to point fashion, the network utilization increases. The server CPU is also charged with the network preparation processing in the case of ODM, but the amount of data is significantly less and yields smaller network utilization. However, the combined overhead of the BNC CPU network processing and the network is less than the overhead of ODM 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. It maintains logs 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 PNC policy ranges between 0.91 and 0.94 for more than 30 clients. This forces the throughput curve to come considerably lower than that of the 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 these policies goes in processing the catalog pages. More specifically, BWC spends 73.00% of its busy CPU time processing catalog pages and PWC 61.90%.

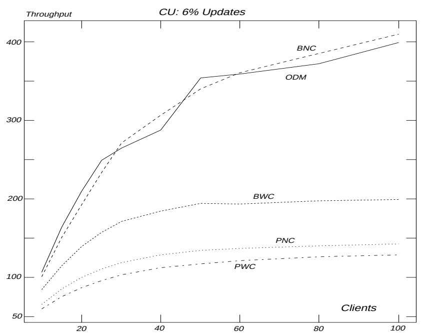

Figure 3: CU Experiment with 6% Update Jobs

For more than 80 clients, PWC offers slightly improved throughput rates over BWC. PWC uses heavily its disk unit (to retrieve portions of the logs at regular intervals) which brings its disk utilization at high levels (between 0.95 and 0.97), while the BWC's disk utilization remains limited (between 0.49 and 0.51). Higher disk utilization means that while the CPU is processing either updates or network related requests, the disk manager forwards into the buffer area the appropriate logs portions that need transmission. This is the reason why PWC offers better throughput rates.

Figure 3 shows the results of the CU experiment with writers updating 6% of the server relation pages. BNC and ODM curves come very close since the benefits and penalties of each of both under the current size of updates provide almost equivalent throughput rates. Essentially, the higher network utilization along with the higher CPU server utilization become equivalent to the high disk utilization of the ODM. PNC and PWC drop below the BWC curve predominantly because the size of the log increases creating more disk accesses for both types of periodic propagation.

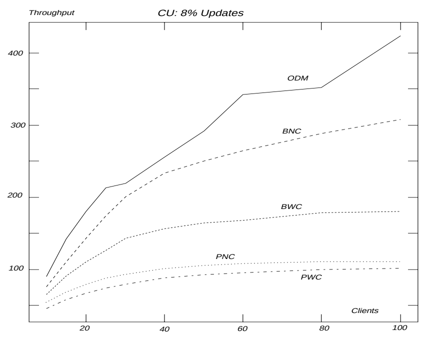

Figure 4: CU Experiment with 8% Update Jobs

Figure 4 depicts the results for the CU experiment with 8% page update 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 forwarding selectively 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 indicates that heavy updates are handled better with directory–based techniques (BWC, PWC).

6.2 VU Experiment

Figure 5 shows the results of the VU experiment for 2% updates in all five propagation configurations. ODM dominates up to 30 clients but then drops below the throughput rates achieved by BNC. 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 indicating that the number of updates–that increases with the number of participating client–makes the server disk resource the main bottleneck point. 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.

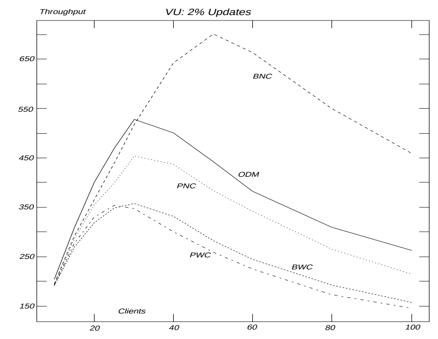

Figure 5: VU Experiment with 2% Update Jobs

PNC achieves lower rates than ODM mostly 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 server CPU for clusters that have more than 25 clients attached (CPU utilization is between 0.47 and 0.83), while PWC demonstrates 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.

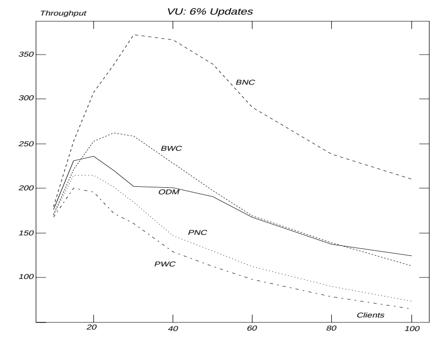

Figure 6: VU Experiment with 6% Update Jobs

Figure 6 shows the results of the experiment with updates of 6%. 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). The latter is the reason for the early decline of the ODM configuration curve. Periodic type of propagation policies suffer also from very high disk utilization rates for more than 15 clients. Similar trends were found for the 8% update selectivity experiment where the various curves become more distinguished than in Figure 6. It is worth mentioning that in this VU workload, where the number of update jobs increases with the number of clients increases, the coupling of a fast server CPU with a fast network (where all 10Mbits/sec are effectively used to transfer modified tuples between the server and the clients) makes broadcasting a more effective way of propagating changes than the lazy and on demand strategy. ODM has to spend considerable amount of time in the disk resident log.

6.3 Experiments with Think Time

To examine the behavior of the various propagation policies in the presence of non–zero think time we re–run the experiments with average client think time 15 secs. For brevity, we present only four of the produced graphs namely those corresponding to experiments CU and VU and for update jobs with page selectivity of 2% and 6%. Figures 7 and 8 depict the results for the CU experiment. ODM is doing better than any other configuration since

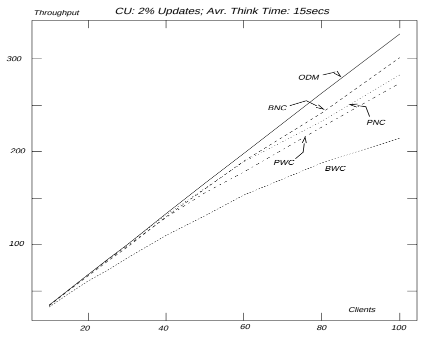

Figure 7: CU, 2% Updates, and Think Time

the think time provides lighter server resource contention. In Figure 8 and for the range 80–100 clients, ODM offers inferior throughput rates than BNC due to the high disk utilization (ranges 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 (Figure 8–space between 80 and 100 clients). The configurations based on the

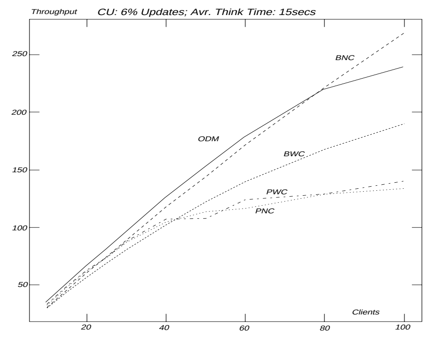

Figure 8: CU, 6% Updates, and Think Time

periodic type of update propagation perform well and their performance approaches that of ODM in Figure 7. They use the server idle periods (implicitly provided by the client think time) to propagate updates. However, under larger updates (i.e. Figure 8) the gap between PNC/PWC and ODM becomes larger since these idle server periods become shorter. Note also that in Figure 8 and between 0–25 clients PNC/PWC give better results than BNC/BWC due to light server resource utilization.

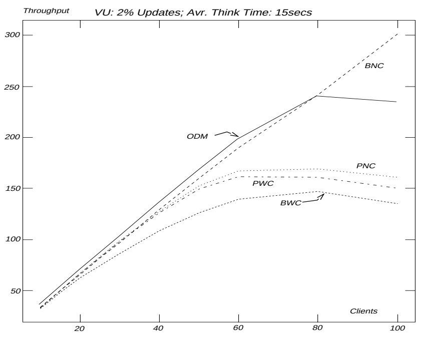

Figure 9: VU, 2% Updates, and Think Time

Figures 9 and 10 show the results of the VU with 2% and 6% update jobs and an average client think time of 15 secs. In Figure 9, the ODM offers the best performance between 0 and 80 clients. Beyond 80 clients, its throughput rates are worse than those of BNC due to heavy server resource utilization. Simple updated tuple broadcasting does relatively well at the beginning of the client space but 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). The BWC configuration gives the poorest rates. The number of binding information pages that have to be retrieved in order to process the updates increases linearly with the number of submitting streams (VU experiment). This contributes significantly to the deterioration of the throughput rates.

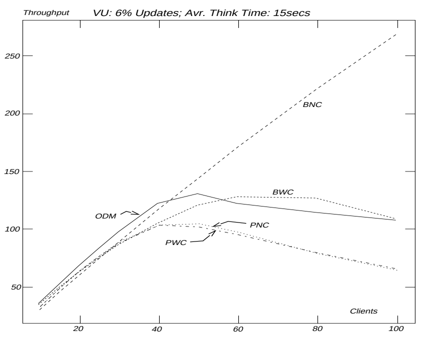

Figure 10: VU, 6% Updates, and Think Time

When the updates become larger (Figure 10), the decline of ODM over BNC comes much earlier –at around 50 clients– since the server log manager copes with larger items of updated data. The server disk utilization varies

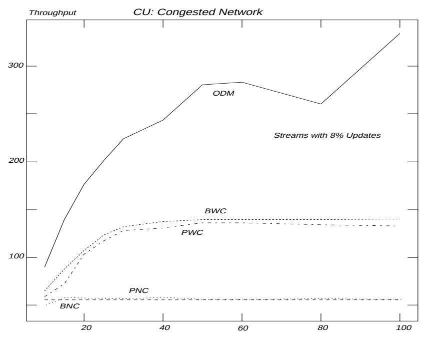

Figure 11: CU, Loaded Network, 8% Updates

between 0.69 and 0.89. BWC behaves better than ODM in the range between 60 and 90 clients since both its main server resources remain moderately loaded (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 and they both provide similar throughput rates. PNC/PWC present the poorest performance. They not only use the log manager heavily but they also utilize the server's CPU very much.

6.4 Sensitivity Analysis

In this last part, we examine the behavior of the five update propagation strategies under diverse network and catalog paging parameter settings. Figure 11 depicts the results of the CU experiment, with job updates of 8% in a highly loaded network. We simulate this congested network by setting the *init_time* to establish a connection between any two machines at 20 msecs and bringing the effective transfer rate over the network at 0.5 Mbits/sec. ODM policy offers the best throughput rates while the configurations based on filtering results after consulting catalog pages (BWC and PWC) are coming second. ODM maintains performance of 1.8 times (on the average) better than PWC and 4.14 times better than BNC throughout the range of the clients 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).

Figure 12 shows the results for the VU experiment with 8% update jobs in a highly loaded network. BWC offers better rates than ODM in the range between 10 and 40 clients since the almost immediately appeared high disk utilization of ODM creates delays that –in the case of BWC– are offset by the network. Nevertheless, for more that 40 clients BWC experiences high network utilization (greater than 0.97). This works negatively for this configuration 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.

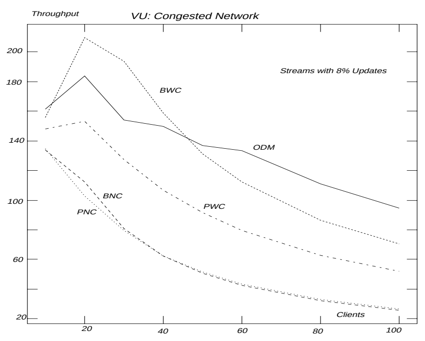

Figure 12: VU, Loaded Network,8% Updates

Figure 13 shows the results of the CU experiment for updates of 2% in a long haul network. Clients and server communicate through dedicated telephone lines at 19,600 BPS. ODM offers the best throughput rates since it uses its incremental log processing. The network bandwidth is almost fully utilized for all configurations. Therefore, 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 forward to the network, they fail to service clients individually creating longer completion times for the submitted streams than those achieved in ODM.

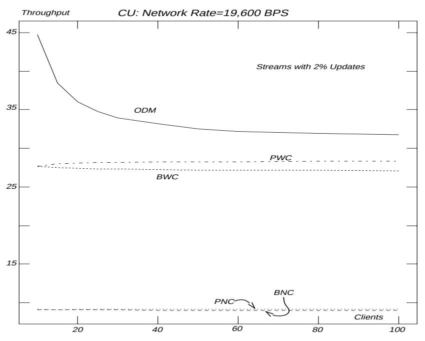

Figure 13: CU, Net. Rate 19,600 BPS

Finally, Figure 14 presents the results for the VU experiments, where light penalties for the catalog based operations of the broadcasting configurations are assumed. More specifically, the server makes one disk access to retrieve the binding conditions for a group of five clients in average and each such page is processed for 20 msecs once in the buffer area. BWC and PWC have come much closer to the ODM which maintains the best overall performance. High network utilization works as an impediment for achieving even higher performance rates in both BWC and PWC for more than 40 clients.

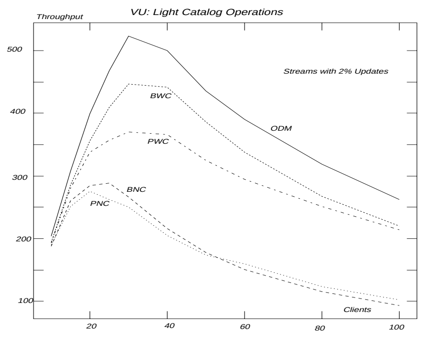

Figure 14: VU, Inexpensive Catalog Access

7 Conclusions

We have proposed server update propagation techniques for the Enhanced Client–Server DBMS architecture and evaluated them under multiple jobs 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), BNC/BWC (broadcasting with/without catalog bindings), and PNC/PWC (periodic broadcasting with/without 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 increases per server. Based on closed network queueing models, we developed software packages for all the strategies in discussion.

Our main experimental results are:

- ODM offers the best performance if none of the server resources reaches full utilization.
- Under high utilization of server resources, the BNC configuration surprisingly offers the best performance when: 1) The updates have small update page selectivities. 2) The number of clients is large (more than 60-70) in the CU family of experiments. 3) 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.
- If ECS operates under a heavily loaded network, then ODM policy provides the best performance independent of the workload. The gains become greater for the more heavily updating curves. This is true if ECS is to function in long haul networks as well.
- 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.

- Periodic type of update propagation demonstrates significant gains when there is non–zero think time. The highest gains were attained for the light update curves.

References

[1] R. Alonso, D. Barbara, and H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM–Transactions on Database Systems*, 15(3):359–384, September 1990.

[2] J. Archibald and J.L. Baer. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model. *ACM–Transactions on Computer Systems*, 4(4), November 1986.

[3] F. Bancilhon, C. Delobel, and P. Kanelakis, editors. *Building An Object–Oriented Database System: The Story of O_2*. Morgan–Kaufmann, San Mateo, CA, 1992.

[4] M. Bellew, M. Hsu, and V. Tam. Update Propagation in Distributed Memory Hierarchy. In *Proceedings of the 6th International Conference on Data Engineering*, pages 521–528, Los Angeles, CA, 1990. IEEE.

[5] J. Bloomer. *Power Programming with RPC*. O'Reilly and Associates, Sebastopol, CA, 1992.

[6] M. Carey, M. Franklin, M. Livny, and E. Shekita. Data Caching Tradeoffs in Client–Server DBMS Architecture. In *ACM–SIGMOD–Conference on the Management of Data*, May 1991.

[7] A. Delis and N. Roussopoulos. Performance and Scalability of Client–Server Database Architectures. In *Proceedings of the 19th International Conference on Very Large Databases*, Vancouver, BC, Canada, August 1992.

[8] A. Delis and N. Roussopoulos. Performance Comparison of Three Modern DBMS Architectures. *IEEE–Transactions on Software Engineering*, 19(2):120–138, February 1993.

[9] A. Delis and N. Roussopoulos. Handling updates in the Enhanced Client–Server DBMS. Technical report, School of Information Systems, Queensland University of Technology, Brisbane, QLD, Australia, March, 1994.

[10] U. Deppisch and V. Obermeit. Tight Database Coperation in a Server–Workstation Environment. In *Proceedings of the 7th IEEE International Conference on Distributed Computing Systems*, pages 416–423, June 1987.

[11] M. Franklin, M. Carey, and M. Livny. Local Disk Caching in Client–Server Database Systems. In *Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, Ireland, August 1993.

[12] W. Kim, J. Garza, N. Ballou, and D. Woelk. Architecture of the Orion Next–Generation Database System. *IEEE–Transactions on Knowledge and Data Engineering*, 2(1):109–124, March 1990.

[13] K. Küspert, P. Dadam, and J. Gunauer. Cooperative Object Buffer Management in the Advanced Information Management Prototype. In *Proceedings of the 13th Very Large Data Bases Conference*, Brighton, UK, 1987.

[14] D. Marakoff and D. Eager. Disk Cache Performance for Distributed Systems. In *Proceedings of the 10th IEEE International Conference on Distributed Computing Systems*, pages 212–219, Paris, France, May 1990.

[15] D. McGovern and C.J. Date. *A guide to SYBASE and SQL Server*. Addison–Wesley, Reading, MA, 1992.

[16] A. Nakamura and M. Takizawa. Reliable broadcast protocol for selectively ordering pdus. In *Proc. of the 11th IEEE Int'l Conf. on Distributed Computing Systems*, 1991.

[17] T. Ng. Propagating Updates in a Highly Replicated Database. In *Proceedings of the 6th International Conference on Data Engineering*, pages 529–536, Los Angeles, CA, 1990. IEEE.

[18] B. Nitzberg and V. Lo. Distributed Shared Memory: A Survey of Issues and Algorithms. *Computer*, 24(8):522–60, August 1991.

[19] S. Ough and R. Sonnier. Spotlight on FDDI. *Unix Review*, 10(10):40–49, October 1992.

[20] N. Roussopoulos. The Incremental Access Method of View Cache: Concept, Algorithms, and Cost Analysis. *ACM–Transactions on Database Systems*, 16(3):535–563, September 1991.

[21] M. Schroeder and M. Burrows. Performance of Firefly RPC. *ACM–Transactions on Computer Systems*, 8(1):1–17, February 1990.

[22] A. Segev and J. Park. Maintaining Materialized Views in Distributed Databases. In *Proceedings of the 5th International Conference on Data Engineering*, pages 262–270, Los Angeles, CA, 1989. IEEE.

[23] A. Stamenas. High Performance Incremental Relational Databases. Master's thesis, University of Maryland, College Park, MD, 1989. Department of Computer Science.

[24] R. Stevens. *Unix Networking Programming*. Prentice Hall, 1990.

[25] L. Svobodova. File Servers for Network–Based Distributed Systems. *Computing Surveys*, 16(4):353–398, December 1984.

[26] Y. Wang and L. Rowe. Cache Consistency and Concurrency Control in a Client/Server DBMS Architecture. In *Proccedings of the 1991 ACM SIGMOD International Conference*, Denver, CO, May 1991.