Cache Management Protocols Based on Re-ordering for Distributed Systems

SungHo Cho[1] and Kyoung Yul Bae[2]

[1] Dept. of Information Science & Telecommunication, Hanshin University, Korea
zoch@hs.ac.kr

[2] Division of Computer Software, SangMyung University, Korea
jbae@smu.ac.kr

Abstract. Database systems have used a client-server computing model to support shared data in distributed systems such as Web systems. To reduce server bottlenecks, each client may have its own cache for later reuse. This paper suggests an efficient cache consistency protocol based on a optimistic approach. The main characteristic of our scheme is that some transactions that read stale data items can not be aborted, because it adopts a re-ordering mechanism to enhance the performance. This paper presents a simulation-based analysis on the performance of our scheme with other well-known protocols. The analysis was executed under the *Zipf* workload which represents the popularity distribution on the Web. The simulation experiments show that our scheme performs as well as or better than other schemes with low overhead.

1 Introduction

A potential weakness of the the client-server model is server bottlenecks that can arise due to the volume of data requested by clients[1,2]. To reduce this problem, each client may have its own cache to maintain some portion of data for later reuse [3,4]. When client caching is used, there should be a transactional cache consistency protocol between client and server to ensure that the client cache remains consistent with the shared data by use of transaction semantics [5,6].

In the literature, many transactional cache consistency algorithms have been proposed. The previous studies[6–8] indicate that Optimistic Two-Phase Locking(O2PL) performs as well as or better than the other approaches for most workloads, because it exploits client caching well and also has relatively lower network bandwidth requirements.

In this paper, we suggests an optimistic protocol called RCP(Re-orderable Cache Protocol). Compared with O2PL, the main advantage of RCP is to reduce

* This work was supported by Ministry of Education and Human Resources Development through Embedded Software Open Resource Center(ESORC) at SangMyung University.

unnecessary operations based on re-ordering. In addition, even if some schemes use multiple-versions to reduces unnecessary operations[1], RCP stores only a single version of each data item for re-ordering. finally, our scheme does not require global deadlock detection which tends to be complex and has frequently been shown to be incorrect.

This paper presents a simulation-based analysis on the performance of some schemes with the *Zipf* distribution which is governed by *Zipf*'s law[9]. So far, a number of groups have shown that, by examining the access logs for different Web servers, the popularity distribution for files on the Web is skewed following *Zipf*'s law. In [10], the authors showed that the *Zipf*'s distribution is strongly applied even to WWW documents serviced by Web servers. Our simulation experiments show that RCP performs as well as or better than other schemes with low overhead.

2 Re-orderable Caching Consistency Protocol

The RCP protocol is one of the avoidance-based algorithms, and it defers write intention declarations until the end of a transaction's execution phase. The difference between RCP and O2PL is that RCP uses a preemptive policy. If possible, RCP tries to re-order transactions which accessed read-write conflicting data items.

Basically, RCP uses the *current version check* for validation. For this, the server maintains read time-stamp $D.T^r$ and write time-stamp $D.T^w$ for each persistent data item. The time-stamps are the time-stamps of the youngest (i.e., lastest in time) committed transaction. When a transaction wants to read a data D_i, the server sends the data with current write time-stamp $D_i.T^w$ if data D_i is not in the local cache of the client. Clients also maintain data and their current time-stamps in the local cache. Hence, a transaction views each data item as a (name, version) pair. Note that we leave the granularity of logical data unspecified in this paper; in practice, they may be pages, objects, etc.

The client of transaction X maintains the following information for the transaction.

- **Set** S_X^R - Set of the data read by transaction X
- **Set** S_X^W - Set of the data written by transaction X
- **Set** S_X^I - Set of the data invalidated by the server during transaction X is running.
- **Time-stamp** T_X^L - The maximum time-stamp among the time-stamps of the data read by transaction X
- **Time-stamp** T_X^U - The minimum time-stamp among the time-stamps of the committing transactions that are conflicted with transaction X (during transaction X is running)

Consider the example in Section 1. If transaction Y is back-shifted, what is the valid time-stamp interval of transaction Y for re-ordering? Intuitively, a back-shifted time-stamp of transaction Y has to be larger than the maximum time-stamp among time-stamps of read data and has to be smaller than the minimum time-stamp among the time-stamps of conflicting and committing transactions. Based on the property, time-stamps T^L and T^U denote the lower-bound and upper-bound of valid interval for re-ordering, respectively. The initial values of time-stamps T^L and T^U are the smallest time-stamp in the system. Sets S^R and S^W are maintained for validation as in optimistic concurrency control. Set S^I is maintained to reduce unnecessary operations.

When transaction X is ready to enter its commit phase, the client sends to the server a message containing sets S^R_X, S^W_X, time-stamps T^L_X and T^U_X. When the server receives the message, it assigns a unique committing time-stamp T^C_X that is equal to the certification time. After that, the server sends a message to each client that has cached copies of any of the updated data items in set S^W_X. The message contains committing time-stamp T^C_X, sets S^R_X and S^W_X. When a remote client gets the invalidation message, it evicts local copies of the data updated by transaction X, and sends an acknowledgment(ACK) to the server. Once all ACKs have been received, the server sets read time-stamp $D_j.T^r$ for each data D_j in set S^R_X and write time-stamp $D_k.T^w$ for each data D_k in set S^W_X to committing time-stamp T^C_X.

Now, we describe how to update the lower-bound time-stamp T^L. Whenever transaction X reads data D_i, time-stamp T^L_X is compared with the current time-stamp $D_i.T^w$. If time-stamp T^L_X is lower than the time-stamp $D_i.T^w$, then T^L_X is set to $D_i.T^w$.

Next, consider the upper-bound time-stamp T^U. When the client of transaction Y gets the invalidation message issued by transaction X, the client updates time-stamp T^U_Y with the following rules;

- For each data D_i in set S^W_X, if data D_i also is in set S^R_Y, then time-stamp T^U_Y is updated based on the following procedures. Otherwise, the following procedures are ignored.
- For each data D_j in set S^R_X, if data D_j also is in set S^W_Y, the client aborts transaction Y, because transaction Y can not be re-ordered.
- If transaction Y is not yet aborted, time-stamp T^U_Y is compared to time-stamp T^C_X. If time-stamp T^U_Y is the initial value, then it is set to time-stamp T^C_X. Otherwise, time-stamp T^U_Y is set to time-stamp T^C_X if it is larger than time-stamp T^C_X.
- Each data D_k in sets S^W_X and S^R_Y is inserted into set S^I_Y to prevent unnecessary operations.

Whenever time-stamps T^L_Y or T^U_Y is changed, transaction Y is aborted if time-stamp T^U_Y is not the initial value and time-stamp T^L_Y is larger than or equal to time-stamp T^U_Y, because the server can not find any back-shifting time-stamp. In addition, whenever transaction Y tries to write the data in set S^I_Y, transaction Y is also aborted to prevent write-write conflicts.

When transaction Y which has accessed invalidated data is ready to enter its commit phase, the client sends to the server a message containing sets S^R_Y, S^W_Y, time-stamps T_Y^L and T_Y^U. When the server receives the message, if time-stamp T_Y^U is not the initial value, it sets committing time-stamp T_Y^C to T_Y^U - δ (δ is an infinitesimal quantity) for re-ordering instead of assigning a new time-stamp. Note that, instead of a specific value, we use an infinitesimal quantity for the value of δ. This approach has an advantage, because it reserves sufficient interval between time-stamps of committing transaction for accepting re-ordered transactions.

After setting T_Y^C to T_Y^U - δ, the server checks whether the back-shifted time-stamp T_Y^C is always larger than time-stamp $D_i.T^r$ for each data D_i in set S_Y^W (Indirect Conflict Check). If time-stamp T_Y^C does not satisfy the rule, transaction Y is aborted. Otherwise, transaction Y is committed with the back-shifted time-stamp T_Y^C.

In commit processing, the server sends time-stamp T_Y^C, sets S_Y^R and S_Y^W to each client that has cached copies of any of the updated data items by transaction Y. After all ACKs are obtained, for each data D_j in set S_Y^R, the server sets time-stamp $D_j.T^r$ to time-stamp T_Y^C if $D_j.T^r$ is less than T_Y^C. In addition, for each data D_k in set S_Y^W, it sets time-stamp $D_k.T^w$ to time-stamp T_Y^C if $D_k.T^w$ is less than T_Y^C.

3 Simulation Study

3.1 Simulation Model

In this section, we compare the proposed scheme(RCP) with O2PL-Invalidate (O2PL) and Call Back Locking(CBL). Our simulation shows the relative performance and characteristics of these approaches. Our study concentrates here mainly on performance aspects, since we are primarily interested in the relative suitability of the cache protocols. Table 1 describes the parameters used to specify the system resources and overhead.

Our simulation model consists of components that model diskless client workstations and a server machine that are connected over a simple network. A client or server is modeled as a simple processor with a microsecond granularity clock. This clock advances as event running as the processor makes "charges" against it. Charges in this model are specified using instruction count.

The number of clients are assumed to be parameter *No_Client.* This study ran experiments using 1 - 25 clients. Each client consists of a *Buffer Manager* that uses an LRU page replacement policy and a *Client Manager* that coordinates the execution of transactions. A *Resource Manager* provides CPU service and accesses to the network. Each client also has a *Transaction Source* which initiates transaction one-at-a-time at the client site.

Compared with client workstations, the server machine has the following differences. The server's *Resource Manager* manages a disk as well as a CPU, and *Concurrency Control Manager* has the ability to store information about the location of page copies in the system and also manages locks (for O2PL and CBL). Since all transactions originate at client workstations, there is no *Transaction Source* module at the server.

Table 1. *System and Overhead Parameter Setting*

Parameter	**Meaning**	**Setting**
Page_Size	Size of a page	4Kbyte
DB_Size	Size of DB in pages	1250
No_Client	No. of clients	1 to 25
No_Tr	No. of transactions per client	1000
Tr_Size	Size of each transaction	20 page
Write_Prob	Write probability	20%
Ex_Tr	Mean time between transactions	0 Sec.
Ex_Op	Mean time between operations	0 Sec.
Client_CPU	Client CPU power	15 MIPS
Server_CPU	Server CPU power	30 MIPS.
Client_Buf	Per client buffer size	5%, 25% of DB
Server_Buf	Server buffer size	50% of DB
Ave_Disk	Average disk access time.	20 millisecond
Net_Bandwidth	Network bandwidth	8Mbps
Page_Inst	Per page instruction	30K inst.
Fix_Msg_Inst	Fixed no. of inst. per msg	20K inst.
Add_Msg_Inst	No. of added inst. per msg	10K inst. per 4Kb
Control_Msg	Size of a control msg.	256 byte
Lock_Inst	Inst. per lock/unlock	0.3K inst.
Disk_Overhead	CPU overhead to perform I/O	5K inst.
Dead_Lock	Deadlock detection frequency	1 Sec.

Upon completion of one transaction, the *Transaction Source* module submits the next transaction. If a transaction aborts, it is re-submitted. It then begins making all of the same pages accesses over again. Eventually, the transaction may complete. The number of transactions in a client is assumed to be the parameter *No_Tr*. For a precise result, each client executes 1000 transactions in this study. The parameter *Tr_Size* denotes the mean number of operations accessed per transaction. The *Ex_Tr* parameter is the mean think time between client transactions, and the *Ex_Op* parameter is the mean think time between operations in a transaction. To make a high degree of data contention, we set both parameters to 0.

Pages are randomly chosen without replacement from among all of the pages according to the workload model described later. The number of pages in the database is assumed to be parameter *DB_Size*. The parameter *Page_Size* denotes the size of each page. A page access cost (*Page_Inst*) is modeled as the fixed number of instructions. The probability that a page read by a transaction will also be written is determined by the parameter *Write_Prob*.

The parameters *Client_Buf* and *Server_Buf* denote the client buffer size and the server buffer size, respectively. In this study, we assume that each client has a small cache (5% of the active database size) or a large cache (25% of the active database size). The CPU service time corresponds to the CPU MIPS rating and the specific instruction lengths given in Table 1. The simulated CPUs of the system are managed using a two-level priority scheme. System CPU requests, such as those for message and disk handling, are given higher priority than user requests. System CPU requests are handled using FIFO queuing discipline, while a processor-sharing discipline is employed for user requests. The disk has a FIFO queue of requests. The average disk access time is specified as the parameter *Ave_Disk*. The parameter *Disk_Overhead* denotes CPU overhead to access disk.

A simple network model is used in the simulator's *Network Manager* component. The network is modeled as a FIFO server with a specified bandwidth (*Net_Bandwidth*). The network bandwidth is set to 8 Mbits/sec which was chosen to approximate the speed of an Ethernet, reduced slightly to account for bandwidth lost to collisions, etc. The CPU cost for managing the protocol to send or receive a message is modeled as a fixed number of instructions per message(*Fix_Msg_Inst*) plus an additional charge per message byte(*Add_Msg_Inst*). The parameter *Control_Msg* denotes the size of a control message.

Our simulation was executed under the *Zipf* workload are governed by *Zipf*'s law. *Zipf*'s law states that if data items are ordered from most popular to least popular, then the number of references to a data tends to be inverse proportional to its rank. in the *Zipf* workload, the probability of choosing data item D_i ($i =$ 1 to *DB_Size* is proportional to 1 / i. Since the first few items in database are much more likely to be chosen than the last few items, the workload has a high degree of locality per client and very high degree of sharing and data contention among clients.

This study uses total system throughput in committed transactions per second (TPS) as our main metric in order to compare the performance of all schemes. Since we use a closed simulation model, throughput and latency are inversely related: the scheme that has better throughput also has low average latency.

3.2 Experiments and Results

In this section, we present the results from performance experiments. This study uses total system throughput in committed transactions per second (TPS) as our main metric in order to compare the performance of all schemes. Since we use a closed simulation model, throughput and latency are inversely related: the scheme that has better throughput also has low average latency.

Fig. 1 shows the total system throughput with a small cache(5% of DB). There is an extremely high degree of data contention. In this experiment, the proposed algorithm (RCP) performs the best with O2PL performing at a somewhat lower level. CBL has the lowest performance throughout the entire range of client population.

In the range from 1 to 5 clients, the performance of all schemes increases. However, beyond 5 clients, all protocols exhibits a "thrashing" behavior in which the aggregate throughput decreases significantly as clients are added to the system. This phenomenon is related with the abort rate as can be seen in Fig. 3.

Generally, the abort rate of locking system such as CBL is lower than that of optimistic based scheme. However, as Fig. 3 shows, the number of aborts of

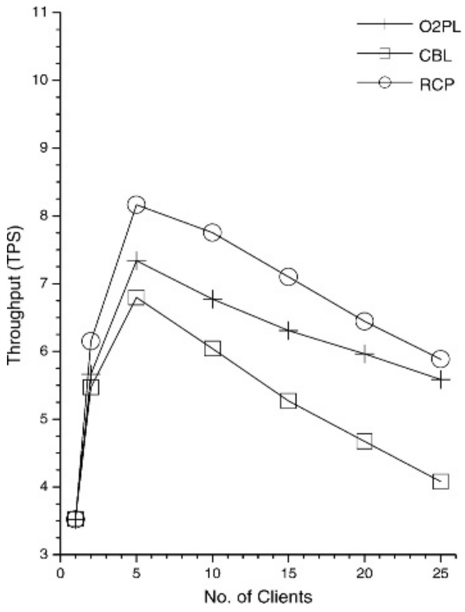

Fig. 1. Throughput (a small cache)

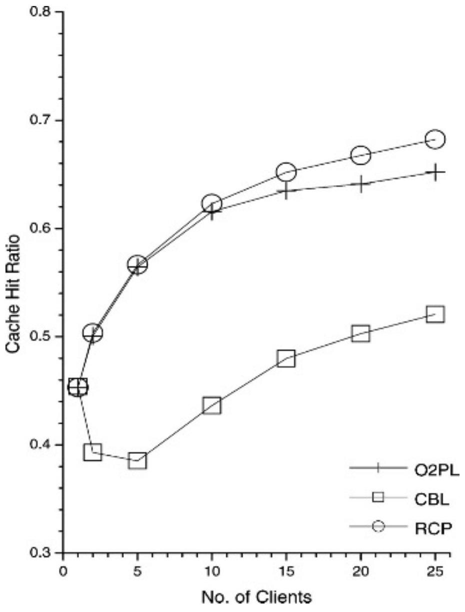

Fig. 2. Cache Hit Ratio (a small cache)

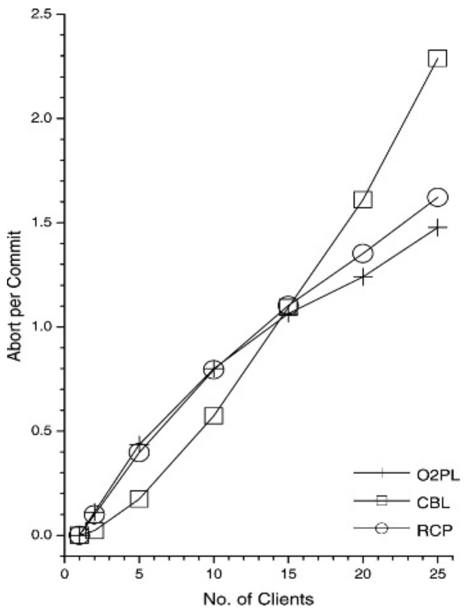

Fig. 3. Number of Aborts (a small cache)

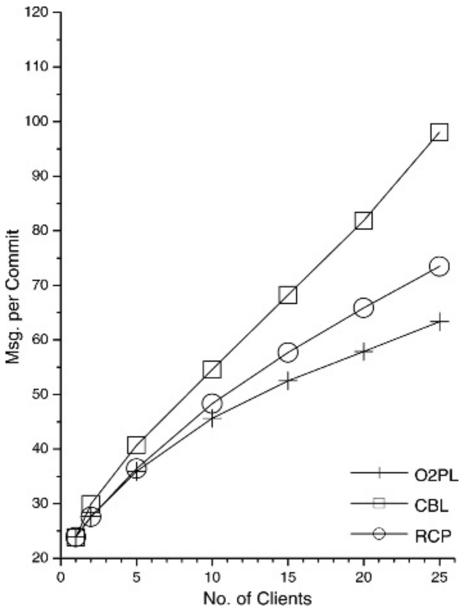

Fig. 4. Message per Commit (a small cache)

CBL exceeds that of other schemes. While write-lock acquisitions are delayed until the end of execution phase in O2PL, write lock declarations occur during the transaction's execution phase in CBL. Hence, in the high client population, CBL generates more aborts than O2PL because its retaining time of write lock is larger than that of O2PL.

Even if the abort rate of RCP is a little higher than that of O2PL, RCP provides the best performance, because it increases data availability by using commit processing without waiting and reduces unnecessary operations by aborting write-write conflicting transactions in their execution phase. In addition, even though RCP does not use any locking method, RCP reduces abort rate significantly compared with CBL, because it re-orders read-write conflicting transactions. Eventually, as Fig. 2 shows, these phenomenons result that the cache hit ratio of RCP is higher than other schemes.

Even if O2PL reduces the abort rates with a lock method, there may be a sudden reduction in the number of active transactions due to transaction blocking. Such blocked transactions eventually leads to a severe degradation in performance, because other transactions which want to access the exclusively locked data are delayed accordingly. In addition, not all the blocked transactions can be committed in O2PL because of deadlock. It makes unnecessary operations.

Re-execution of a transaction is more efficient than its execution in the first phase, because pages that were accessed by the transaction are already available at the client of transaction execution. Note that CBL can not abort conflicting transactions in their execution phase. In contrast, RCP aborts write-

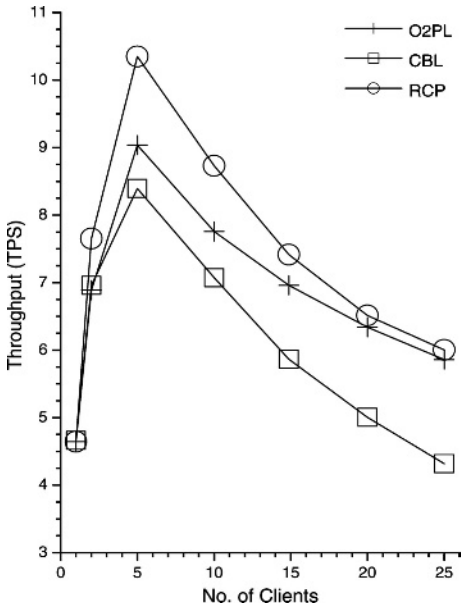

Fig. 5. Throughput (a large cache)

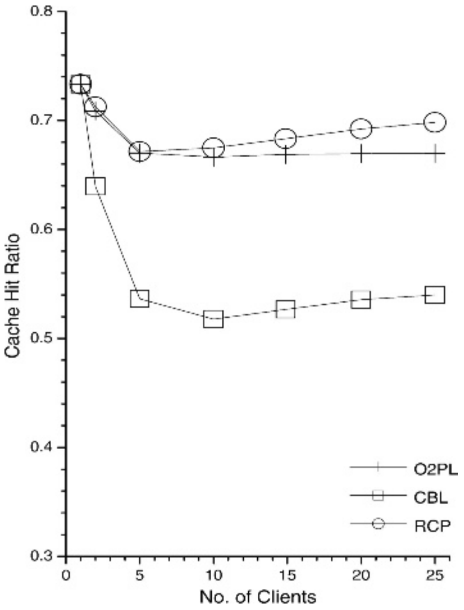

Fig. 6. Cache Hit Ratio (a large cache)

write conflicting transactions in their execution phase. By these reasons, the abort rate has only a small impact on the performance of RCP compared with CBL.

As Fig. 4 represents, CBL sends significantly more messages per commit than other protocols throughout the entire range of client population because of the highest abort rate. Due high message cost and high aborts rate, CBL has the lowest performance throughout in the *Zipf* workload.

Fig. 5 shows the total system throughput with a large (25% of DB) client cache. Compared with Fig. 1, the performance of all protocols slightly increases because of extending buffer size. However, relatively large cache size does not significantly affect the performance of the schemes. It denotes that extremely high number of aborts leads the network and server to bottlenecks.

Fig. 6 represents the cache hit ratio. The ratio of CBL is extremely lower than other schemes. The reason is that the actions that remove the invalidated data items in caches under O2PL and RCP occur only after the server decides a transaction's commit. However, preemptive pages does not guarantee the transaction's commit under CBL. Compared with Fig. 2, Fig. 6 also shows that cache hit ratio is less affected by the restart-induced buffer hits compared with the case of the small cache buffer. The restart-induced buffer hits means that most of all data items needed by re-started transactions were available in the client buffer. This was further borne out by the client hit rate. Hence, it also explains why the cache hit ratio of RCP is higher than that of O2PL.

4 Conclusion

In this paper, we suggested a new cache consistency protocol for client-server database systems which provides serializability. Our scheme is based on an optimistic concurrency control with re-ordering approach. In our scheme, the server tries to re-order some read-write conflicting transactions with low overhead. In addition, the suggested scheme aborts write-write conflicting transactions in their execution phase.

Compared with O2PL, our approach has advantages such as increasing data availability by use of the no-wait commit approach, reducing unnecessary operations by aborting write-write conflicting transactions in their execution phase and eliminating the maintaining cost of lock and deadlock detection algorithm. The deadlock freedom of our protocol considerably simplifies the complexity of an actual implementation. In addition, compared with CBL, our scheme reduces the transaction abort rate and unnecessary operations.

This paper presents the results of simulation experiments with a detailed simulator under the *Zipf* workload. Throughout our simulation experiments, CBL shows the worst performance because it suffers from a high message cost. RCP shows the best performance, because O2PL limits the transaction concurrency level. By the experimental results, we show that our scheme performs as well as or better than the other approaches with low overhead.

References

1. E. Pitoura and P. K. Chrysanthis, "Multiversion Data Broadcast," *IEEE Transacions on Computers*, Vol.51, No.10, pp 1224–1230, 2002.
2. Daniel Barbara, "Mobile Computing and Database - a Survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No.1, pp.108–117, 1999.
3. J. Jing, A. Elmagarmid, A. Helal and R. Alonso, "Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments", *ACM/Baltzer Mobile Networks and Applications*, Vol.2, No.2, 1997.
4. C. F. Fong, C. S. Lui and M. H. Wong, "Quantifying Complexity and Performance Gains of Distributed Caching in a Wireless Network Environment", *Proceedings of the 13th International Conference on Data Engineering*, pp.104–113, April 1997.
5. V. Gottemukkala, E. Omiecinski and U. Ramachandran, "Relaxed Consistency for a Client-Server Database," *Proc. of International Conference on Data Engineering*, February, 1996.
6. A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient optimistic concurrency control using loosely synchronized clocks," *In Proc. of the ACM SIGMOD Conf. on Management of Data*, pp. 23–34, 1995.
7. M. J. Carey, M. J. Franklin, M. Livny, and Shekita, "Data caching tradeoffs in client-server DBMS architectures," *In Proc. of the ACM SIGMOD Conf. on Management of Data*, pp. 357–366, 1991.
8. M. J. Franklin, M. J. Carey, and M. Livny, "Local disk caching in client-server database systems," *In Proc. of the Conf. on Very Large Data Bases (VLDB)*, pp. 543–554, 1993.
9. G.K. Zipf, *Human Behavior and the Principles of Least Effort*, Reading, Mass., Addison Wesley, 1949.
10. V. Almeida, A. Bestavros, M. Crovella, and A. D. Oliveira, "Characterizing reference locality in the WWW," *Proceedings of the 1996 International Conference on Parallel and Distributed Information Systems (PDIS '96)*, pp. 92-103, 1996.