Performance Analysis of Dynamic Finite Versioning Schemes: Storage Cost vs. Obsolescence

Arif Merchant, Kun-Lung Wu, *Member, IEEE*,
Philip S. Yu, *Fellow, IEEE*, and Ming-Syan Chen, *Senior Member, IEEE*

Abstract—Dynamic finite versioning (DFV) schemes are an effective approach to concurrent transaction and query processing, where a finite number of consistent, but maybe slightly out-of-date, *logical* snapshots of the database can be dynamically derived for query access. In DFV, the storage overhead for keeping additional versions of changed data to support the logical snapshots and the amount of obsolescence faced by queries are two major performance issues. In this paper, we analyze the performance of DFV, with emphasis on the trade-offs between the storage cost and obsolescence. We develop analytical models based on a renewal-process approximation to evaluate the performance of DFV using $M \geq 2$ snapshots. Asymptotic closed-form results for high query arrival rates are given for the case of two snapshots. Simulation is used to validate the analytical models and to evaluate the trade-offs between various strategies for advancing snapshots when $M > 2$. The results show that 1) the analytical models match closely with simulation; 2) both the storage cost and obsolescence are sensitive to the snapshot-advancing strategies, especially for $M > 2$ snapshots; and 3) generally speaking, increasing the number of snapshots demonstrates a trade-off between storage overhead and query obsolescence. For cases with skewed access or low update rates, a moderate increase in the number of snapshots beyond two can substantially reduce the obsolescence, while the storage overhead may increase only slightly, or even decrease in some cases. Such a reduction in obsolescence is more significant as the coefficient of variation of the query length distribution becomes larger. Moreover, for very low update rates, a large number of snapshots can be used to reduce the obsolescence to almost zero without increasing the storage overhead.

Index Terms—Multiple versions, transaction processing, concurrent transaction and query processing, analytical modeling, renewal process, two-moment approximation, and dynamic finite versioning.

1 Introduction

Concurrent transaction and query processing has become increasingly important as more and more decision makers are depending on more up-to-the-minute information from their database systems [1]. In general, only transactions can update the database, while queries are read-only actions, which can support decision making. These queries tend to be ad hoc, take a long time to execute, and may not need to access the most up-to-date database. For example, a bank manager may want to know the current total balance of all the checking accounts while customers are concurrently making deposits/withdrawals to/from their accounts. For decision support, this ad hoc query can access slightly obsolete data (say, a consistent database existed 10 minutes ago), and need not compete for account access with other ongoing update transactions initiated by customers.

- *A. Merchant was with IBM Research. He is now with Hewlett Packard Laboratories, MS 14-13, 1501 Page Mill Road, Palo Alto, CA 94303. E-mail: arif@hpl.hp.com.*
- *K.-L. Wu and P.S. Yu are with IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. E-mail: {klwu, psyu}@watson.ibm.com.*
- *M.-S. Chen was with IBM Research. He is now with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. E-mail: mschen@cc.ee.ntu.edu.tw.*

Manuscript received Nov. 3, 1993.
For information on obtaining reprints of this article, please send e-mail to: transkde@computer.org, and reference IEEECS Log Number K96052.

In concurrent transaction and query processing, severe interference due to data contention may exist between transactions and queries if both are to access the same database through conventional concurrency control mechanisms, such as two-phase locking [2], [3], [4]. Three alternative approaches can be used to address this issue. The first approach compromises serializability for queries, while the other two approaches provide serializable execution for them. One example of the first approach is the *cursor stability* used in IBM's DB2 [5] where the read locks are given up as the cursor moves from one page to the next, potentially causing queries to read inconsistent data. Another example of compromising serializability is the notion of *epsilon serializability* where bounded inconsistency is allowed for queries [6], [7].

The second approach maintains two physically separate databases, one for transaction and the other for queries [1]. Storage is doubled and extra costs are required to periodically refresh data from the transaction database to the query database. As a result, queries may access a substantially obsolete database due to the potentially high costs associated with each refreshing.

The third approach maintains multiple versions of a data item when it is updated, and is generally called *multiversioning* [8], [9] (or *transient versioning* in [1]). Interference between transactions and queries can be eliminated or re-

1041-4347/96$05.00 ©1996 IEEE

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

duced. Typically, depending on the total number of versions maintained for each data item, most existing multiversioning schemes can be further classified into two categories: two versions and an unrestricted number of versions. Two-version schemes [10] reduce, but do not eliminate, interference because only one old version is maintained. On the other hand, at the expense of high storage cost and complex version management, schemes maintaining an unrestricted number of versions can eliminate interference [11], [4], [12], [13], [14], [15].

However, the increases in storage overhead and version-management complexity, including garbage collection, may become significant when an unrestricted number of versions are maintained. Recently, different versioning schemes have been proposed to address these problems by maintaining only a fixed number of versions [16], [17], [18], [19], [20]. A general class of such versioning schemes, referred to as *dynamic finite versioning* (DFV), which allows the STEAL policy by transactions [21][1] and noninterfering snapshot advancements, has been proposed to effectively support concurrent transaction and query processing [19].

In DFV, transactions access the most up-to-date data in the database, and at least two consistent *logical* database snapshots, called query snapshots (QSs), are dynamically maintained for query access. A concurrency control mechanism, such as two-phase locking, is assumed to maintain a serializable order for different transactions. However, there is no synchronization between transactions and queries, or among different queries. Any read-only transaction that requires the most up-to-date information can access the database through the concurrency control mechanism. Taking a snapshot in DFV only captures the updates from the most recently committed transactions since the last snapshot; it does not physically copy the entire database. The updates reflected in each snapshot determine the storage overhead. In general, queries read more recent data if more snapshots are maintained. However, depending on the workload environment, there can be a trade-off between the average storage overhead and query obsolescence, a measure of how out-of-date the data that a query reads are.

In this paper, we develop analytical models to evaluate the effectiveness of DFV, with emphasis on understanding the trade-offs between the average storage overhead and query obsolescence. In the case of two query snapshots, we derive an approximation for the intersnapshot interval using a distribution based on the mean value of this interval (a one-moment approximation). The process of snapshot advances is then modeled approximately as a renewal process, and formulae are derived for the mean storage cost, mean obsolescence and the mean number of updates missed by a query. Closed-form asymptotic results are given for these performance measures when the query arrival rates are high. In the case of more than two snapshots, we approximate the inter-snapshot interval using a distribution based on both the mean and the variance of this interval (a two-moment approximation), and derive a set of recurrence relations for the two moments. A fixed point approach is used to deduce the values of the moments, and the query snapshot advance process is again treated as a renewal process to derive the values of the performance metrics. Simulation is used to validate the analytical models and to examine the trade-offs among different strategies for advancing query snapshots when a large number of QSs are used. The results show that the analytical models match closely with simulation, and both the storage overhead and obsolescence can be sensitive to the QS-advancing strategies, especially for $M > 2$ QSs. Contrary to conventional wisdom, we found that increasing the number of snapshots to reduce obsolescence may not increase the storage cost. The study shows various factors affecting the appropriate number of snapshots to choose.

The paper is organized as follows. In Section 2, we describe DFV schemes, with emphasis on the snapshot advance mechanisms, and the performance issues studied in the paper. In Section 3, the analytical models are presented. Validation of the analytical results and performance evaluation of various snapshot-advance policies and the trade-offs between storage cost and obsolescence are then presented in Section 4.

2 Dynamic Finite Versioning Schemes

In this section, we present a brief description of the DFV schemes and some of the issues we investigate in the rest of this paper. More details concerning other implementation issues, such as storage management and version identification, can be found in [19]. In DFV, a finite number of *logical* versions for each page is dynamically maintained. Note that it is the distinct, physical versions of each page that determine the storage overhead, not the logical ones. From these versions, up to M consistent database snapshots can be derived for queries to read, and the most up-to-date data in the database are available for transactions to access without lock contention from queries.

2.1 DFVs with Two Snapshots

For simplicity, we start by assuming that the non-STEAL policy is used, i.e., uncommitted updates are not allowed to be written into the database, and describe DFVs with two logical snapshots, QS_0, the old snapshot, and QS_1, the recent one. (DFV schemes that can handle the STEAL policy are presented later in Section 2.3.) Each logical snapshot QS_i, $i = 0$ or 1, consists of a time-stamp t_i (representing the time QS_i was taken), a list QL_i of active queries accessing the snapshot, and the latest updates committed since the previous snapshot. Besides the two logical snapshot versions, the updates *committed* since the most recent snapshot are separately maintained for transaction access. Thus, transactions may access the latest data without contention from queries. These newly committed updates may become part of a new snapshot, which may be taken at any time without quiescing any transaction or query.

1. In the STEAL policy [21], dirty pages updated by active transactions in the buffer are allowed to be written back into the database before the end-of-transaction (EOT) statement. On the other hand, in the non-STEAL policy, dirty pages updated by a transaction must be buffered and cannot be written back to the database on disk until the transaction commits.

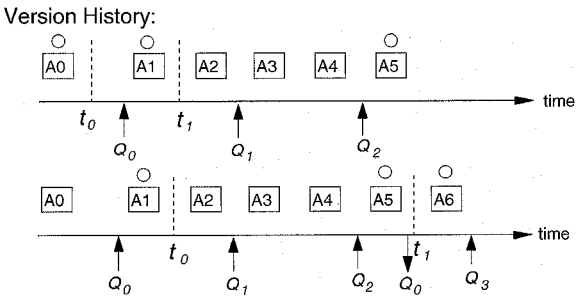
Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

Arriving queries access the recent snapshot QS_1, and join the active query list QL_1. When a query completes, it is removed from the active query list of its snapshot. When the last query accessing the old snapshot QS_0 completes, a query snapshot advancement (QS-advance) can be triggered as follows:

$$QS_0 \leftarrow QS_1 \ ; \ QS_1 \leftarrow \text{new snapshot}.$$

The active queries that arrived before this QS-advance, and were accessing QS_1 continue to access the same snapshot, which is now QS_0. These still-active queries form the new QL_0. The next QS-advance will take place when the last of these completes. However, what if there were no active queries accessing QS_1 before a snapshot advance? Since the new QL_0 starts empty, there will be no completions to trigger the next snapshot advance. As a result, we insert a "system-scheduled" snapshot advance at time Δ in the future.

As an example, Fig. 1 shows a scenario for a query snapshot advance, a version history of a page and the different versions that queries will access. Different versions of page A, denoted by $A0$, ..., $A6$, were created at different times. Two snapshots with timestamps of t_0 and t_1, respectively, are maintained. Q_0, ..., Q_3 denote different queries, starting at different times. Q_0 was started after t_0 but before $A1$ was created; Q_1 and Q_2 both were started after t_1 but at different times. In DFV, queries Q_1 and Q_2 both read version $A1$. However, they would have access $A2$ and $A4$, respectively, in a traditional multiversioning approach. In DFV, a QS-advance is automatically initiated at the instant when all active queries accessing the old snapshot is completed. Compared with traditional multiversioning schemes in which the entire version history may have to be maintained for potential query access (unrestricted number of versions), DFV only keeps three versions (those marked with a circle), and allows query snapshots to advance automatically. Note that we do not include the uncommitted version in Fig. 1.



Version Access:

	Traditional multiversioning	Dynamic finite versioning
Q_0	A0	A0
Q_1	A2	A1
Q_2	A4	A1
Q_3	A6	A5

Fig. 1. A QS-advance scenario and version access (↑: query arrival, ↓: query completion).

2.2 Major Issues with More Than Two Snapshots

The two-snapshot DFV can be easily extended to use more than two snapshots. The simplest extension is to keep M snapshots QS_0, QS_1, ..., QS_{M-1}, where QS_0 is the oldest snapshot, QS_1 the next oldest, and so on, and to trigger a snapshot advance when the last active query accessing QS_0 completes. This can be described as follows:

$$\text{for } j = 0 \text{ to } M-2\{QS_j \leftarrow QS_{j+1}\}$$
$$QS_{M-1} \leftarrow \text{new snapshot}.$$

It is clear that using more snapshots reduces the obsolescence in the data seen by queries. For instance, if a snapshot were taken immediately after every query arrival, queries would always see the most up-to-date data, and there would be no obsolescence. However, this is at the expense of a higher storage overhead, since more of the committed updates would be stored in the snapshots. This trade-off is important, especially in determining the optimal number of snapshots to be kept, and we shall examine it in the following sections.

There are several ways of controlling the snapshot advance process other than the simple one described above. In a different completion-triggered approach, a snapshot advance could take place whenever *any* one of the QL_i becomes empty upon a query completion. We call this the "completion-triggered, any QL_i empty" policy. Besides, at the time when a query completes, a new snapshot can be taken at the moment when a query arrives. In an arrival-triggered policy, a snapshot advance takes place whenever an arriving query finds a QL_i empty; thus, such a query sees no obsolescence. A generalization of this approach is to allow a snapshot to advance only if there have been at least $K \geq 1$ queries accessing QS_{M-1} and there is some empty QL_i upon a query arrival. While the above simple case is our base case, we also explore the variations described here.

2.3 DFVs with STEAL Policy

If the STEAL policy is allowed, it is possible that there are uncommitted page versions in the database. With the existence of uncommitted versions potentially scattered around in the database, it can be difficult to efficiently establish a transaction-consistent logical snapshot at any moment without quiescing the transaction processing.

In DFV, a unique design is provided so that the STEAL policy is allowed and a logical snapshot can be taken efficiently at any time [19]. To facilitate the non-interfering snapshot-taking, associated with each physical version of a page are two variables: *tid*, transaction identifier updating this page, and t_u the time when the page is updated. In addition, a list $AcTL_c$ of currently active transactions is kept. When a transaction is started, its unique identifier *tid* is stored into $AcTL_c$, and is removed from $AcTL_c$ when it is committed. Removing a *tid* from $AcTL_c$ is equivalent to implicitly making the uncommitted version into the latest committed version. Taking a logical, consistent snapshot now becomes as simple as making a copy of $AcTL_c$ and recording the current time. Thus the action of a QS-advance in DFV with two snapshots can be accomplished as follows:

$$QL_0 \leftarrow QL_1; \quad AcTL_0 \leftarrow AcTL_1; \quad t_0 \leftarrow t_1;$$
$$QL_1 \leftarrow \text{empty}; \quad ACTL_1 \leftarrow ACTL_c; \quad t1 \leftarrow \text{current time},$$

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

where $AcTL_i$, $i = 0$ or 1, is the list of active transactions at time t_i. Notice that the manipulation of these data structures is the only action required for a QS-advance and is done without any disk access to the database; no quiescing of either the transaction or query processing is required.

When a transaction writes to a page in the database, it does not overwrite the latest committed version of the page with the new data. Instead it *creates* a new version by overwriting an old version, if available, that is not a snapshot version [19]. However, an uncommitted version of a page can be written multiple times by the same transaction.

Together with t_i and $AcTL_i$, variables *tid* and t_u are also used to identify correct snapshot versions for query access. Since its *tid* is removed from $AcTL_c$, but not from $AcTL_0$ or $AcTL_1$, when a transaction is committed, the recent snapshot version of a page is the latest version with $t_u < t_1$, $tid \notin ACTL_1$, and $tid \notin ACTL_c$, which represents the latest version committed before t_1. Similarly, the old snapshot version of a page is the latest version with $t_u < t_0$, $tid \notin ACTL_0$, and $tid \notin ACTL_c$, representing the latest version committed before t_0 [19].

Notice that, besides the two logical snapshot versions, only the latest committed and uncommitted versions since the most recent snapshot are kept; intermediate versions created are automatically discarded through transaction overwrites, thus reducing the storage cost. Moreover, since transaction execution times are short, our simulation showed that the storage overhead for the uncommitted versions is negligible. Namely, the impact of the STEAL policy on storage overhead is negligible. Thus, in the following analyses of storage cost, we only focus on the committed versions.

3 Analytical Modeling

We construct analytical models of DFVs, primarily for the completion triggered case. Let us begin by defining more precisely the model we employ.

We assume that the system supports a maximum of M snapshots at a time. Update transactions for a page P arrive independently of queries according to a Poisson process with rate λ_u. Queries are assumed to arrive according to a Poisson process with rate λ_q, and the time to service a query is exponentially distributed with mean $1/\mu$, all queries being served independently with no limit on the number that can be served simultaneously. (In practice there may be some restrictions on the number of queries served simultaneously due to limitations on resources other than storage capacity. However, we ignore these for two reasons:

1) we are primarily trying to model the effects of storage capacity limitations and DFV policies, and
2) so long as these limitations are not severe, the delay before a query begins execution would be small compared with query execution time, and hence a Poisson process arrival model is still reasonable.

We consider query length distributions other than exponential in Section 4.2.4 using simulation.)

Let

$QS_j(t) \equiv$ the jth query snapshot at time t

$T_i \equiv$ time of the ith snapshot advance

$Y_i \equiv T_i - T_{i-1}$

$y_i \equiv E[Y_i]$

$NQ_j(t) \equiv$ number of queries accessing QS_j at time t

$N_i \equiv NQ_0(T_i +)$.

Recall that there are M snapshots $QS_0(t)$, $QS_1(t)$, ..., $QS_{M-1}(t)$ with $QS_0(t)$ the earliest snapshot and $QS_{M-1}(t)$ the latest; we drop the argument t when the time of reference is obvious. In our base system, a snapshot advance takes place whenever the last query accessing QS_0 completes. However, if the active query list of QS_0 immediately after a snapshot advance is empty, there are no completions to trigger a snapshot advance, and we must insert a "system-scheduled" snapshot advance at time $T_i + \Delta_{i+1}$. The value of Δ_i can have a significant effect on system performance, as we shall see.

The performance metrics of interest measure the obsolescence in the data seen by queries and the number of versions stored per page. For simplicity, we shall only consider a single page P, and later show how to scale the results over the entire database. Let $NCopies(t)$ denote the number of copies of P needed at time t. Then the mean number of versions of P that must be stored is

$$\lim_{t\to\infty} \frac{1}{t} \int_0^t NCopies(x)dx.$$

The obsolescence of P seen by a query Q arriving at time $t \in (T_i, T_{i+1})$ is $T_u - T_i$, where T_u is the time of the last update to P between T_i and t (see Fig. 2). If there are no updates in this interval, the obsolescence is zero. Another measure of obsolescence is the number of updates to P missed by Q, that is, the number of updates to P that arrive in (T_i, t).

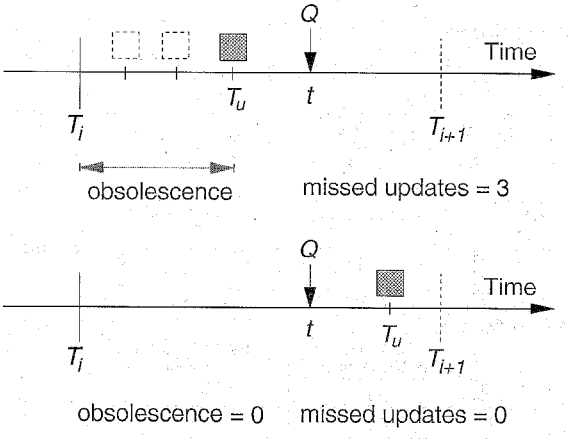


Fig. 2. Obsolescence and updates missed.

3.1 The Case of Two Snapshots

Let us consider first the case $M = 2$, because it is both important and relatively simple. We construct a one-moment approximation for the intersnapshot times Y_i. We shall then deduce the mean values of the performance metrics by approximating the process $T_0, T_1, \ldots$ as a renewal process.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

The number $N(T_i + t)$ of queries active in the current snapshot between snapshot-advance times T_i and T_{i+1}, can be modeled as a birth-death process with a fixed arrival rate λ_q and death rates $\mu_j = j\mu$ (see Fig. 3), starting in state zero. This process is well known as the Telephone-Trunking process, or the M/M/∞ queue, and it can be shown that $N(T_i + t)$ has a Poisson distribution with mean $\lambda_q(1 - e^{-\mu t})/\mu$ (see [22, Eq. 7.9], [23, page 291]). This motivates the one-moment approximation:

$$N_i = NQ_0(T_i +) = NQ_1(T_i -) \approx$$
$$\text{Poisson with mean } \rho_i = \lambda_q\left(1 - e^{-\mu y_i}\right)/\mu. \quad (1)$$

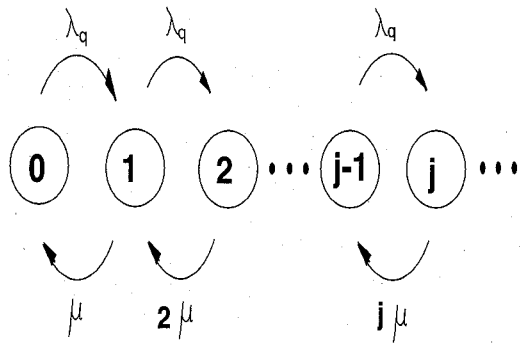


Fig. 3. Telephone trunking process.

Since the query service times are exponentially distributed, we may ignore the service already received by the queries at time T_i, and the remaining service times are distributed exponentially with mean $1/\mu$. Let $R_1, R_2, \ldots$ denote the remaining service times of the queries, so that the length of the next inter-snapshot time is

$$Y_{i+1} = \begin{cases} \max\left(R_1, R_2, \ldots, R_{N_i}\right) & \text{if } N_i > 0 \\ \Delta_{i+1} & \text{if } N_i = 0. \end{cases}$$

$$\begin{aligned} Pr[Y_{i+1} < t] &= \sum_k Pr[Y_{i+1} < t \wedge N_i = k] \\ &= Pr[N_i = 0]Pr[\Delta_{i+1} < t] \\ &\quad + \sum_{k>0} Pr[N_i = k]Pr[R_1 < t]Pr[R_2 < t] \cdots Pr[R_k < t] \\ &= Pr[N_i = 0]Pr[\Delta_{i+1} < t] + \sum_{k>0} Pr[N_i = k]\left(1 - e^{-\mu t}\right)^k \\ &= E\left[\left(1 - e^{-\mu t}\right)^{N_i}\right] - Pr[N_i = 0]Pr[\Delta_{i+1} > t] \quad (2) \\ &= exp\left(-\rho_i e^{-\mu t}\right) - e^{\rho_i} Pr[\Delta_{i+1} > t]. \quad (3) \end{aligned}$$

$$\begin{aligned} y_{i+1} = E[Y_{i+1}] &= \int_0^\infty Pr[Y_{i+1} > t]dt \\ &= \int_0^\infty \left(1 - exp\left(-\rho_i e^{-\mu t}\right)\right)dt + e^{-\rho_i}\int_0^\infty Pr[\Delta_{i+1} > t]dt \\ &= \int_0^{\rho_i} \frac{1 - e^{-t}}{t}dt + e^{-\rho_i}E[\Delta_{i+1}] \\ &= Ei(\rho_i) + \ln(\rho_i) + \gamma + e^{-\rho_i}E[\Delta_{i+1}] \quad (4) \end{aligned}$$

where E_i is the *exponential-integral function* and γ is Euler's constant (see [24]).

As $i \to \infty$, assuming a stationary state is achieved,

$$y_i = y_{i+1} \approx Ei(\rho_i) + \ln(\rho_i) + \gamma + e^{-\rho_i}E[\Delta_{i+1}]$$
$$\text{where } \rho_i = \lambda_q\left(1 - e^{-\mu y_i}\right)/\mu. \quad (5)$$

For each choice of Δ_i, this gives us an equation in y_i. The equation is transcendental, and thus not amenable to exact solution; however, it is easy to solve numerically.

3.1.1 Calculating the Performance Measures

To compute the performance measures, we approximate the inter-snapshot intervals $Y_0, Y_1, \ldots$ as independent random variables identically distributed according to (2) using the stationary value of y_i; let $Y \equiv \lim_{i\to\infty} Y_i$ and $y \equiv E[Y] = \lim_{i\to\infty} y_i$.

The average **number of versions** stored per page is given by

$$\lim_{t\to\infty}\frac{1}{t}\int_0^t NCopies(x)dx = \lim_{i\to\infty}\frac{\sum_{j=0}^{i}\int_{T_j}^{T_{j+1}} NCopies(t)dt}{\sum_{j=0}^{i} Y_{j+1}}$$
$$= \lim_{i\to\infty}\frac{E\left[\int_{T_i}^{T_{i+1}} NCopies(t)dt\right]}{E[Y_{i+1}]}. \quad (6)$$

Besides the copy of a page P in the database, one additional copy of P is stored in QS_1 if there was an update to P in the interval between the snapshots QS_0 and QS_1, and yet another if there was an update after QS_1 was taken. Thus, at time $t \in (T_i, T_{i+1})$, the number of copies is

$$NCopies(t) = 1 + \mathbf{1}(\text{update in } (T_{i-1}, T_i)) + \mathbf{1}(\text{update in } (T_i, t)) \quad (7)$$

where the indicator function $\mathbf{1}(e)$ is one if e is true and zero otherwise. From (7),

$$E\left[\int_{T_i}^{T_{i+1}} NCopies(t)dt\right] = E\left[Y_{i+1} + Y_{i+1}\left(1 - e^{-\lambda_u Y_i}\right) + \int_{T_i}^{T_{i+1}}\left(1 - e^{-\lambda_u(t-T_i)}\right)dt\right].$$

Substituting in (6),

$$\lim_{t\to\infty}\frac{1}{t}\int_0^t NCopies(x)dx \approx 3 - \frac{1}{\lambda_u y} - \left(1 - \frac{1}{\lambda_u y}\right)E\left[e^{-\lambda_u Y}\right]. \quad (8)$$

Using the distribution (2) and simplifying, we find

$$E\left[e^{-\lambda_u Y}\right] = \rho^{-\lambda_u/\mu}\int_0^\rho x^{\lambda_u/\mu}e^{-x}dx + e^{-\rho}E\left[e^{-\lambda_u\Delta}\right] \quad (9)$$
$$= \rho^{-\lambda_u/\mu}\Gamma_\rho(\lambda_u/\mu + 1) + e^{-\rho}E\left[e^{-\lambda_u\Delta}\right], \quad (10)$$

in terms of the *incomplete gamma function* $\Gamma\rho$ (see [24], [25, p. 167]).

The **number of updates missed** by a query (see Fig. 2) that arrives c seconds after the last snapshot is distributed as a Poisson random variable with mean $\lambda_u c$. We must, therefore, look at the distribution of the "current life" C of a snapshot, or the time elapsed since the last snapshot advance. Since we have assumed $Y_0, Y_1, \ldots$ to be independent and identically distributed, $\{T_0, T_1, \ldots\}$ forms a renewal

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

process, and therefore the distribution of current life is given by (see [26, Sec. 5.6])

$$F_C(t) \equiv Pr[C < t] = \frac{1}{y}\int_0^t \big(Pr[Y > x]\big)dx \tag{11}$$

$$f_C(t) \equiv F_C'(t) = Pr[Y > t]/y. \tag{12}$$

Using this, the expected value of a function of C can be expressed directly in terms of the expected value of a function of Y:

$$\begin{aligned} E[g(C)] &= \int_0^\infty g(t) f_C(t)dt \\ &= \frac{1}{y}\int_0^\infty g(t) Pr[Y > t]dt \\ &= \frac{1}{y}\int_0^\infty g(t)\int_t^\infty f_Y(x)dxdt \quad f_Y \text{ is the } p.d.f \text{ of } Y \\ &= \frac{1}{y}\int_{x=0}^\infty f_Y(x)\int_{t=0}^x g(t)dtdx \\ &= \frac{1}{y}E\left[\int_0^Y g(t)dt\right]. \end{aligned} \tag{13}$$

Thus, the mean current life is

$$\begin{aligned} E[C] &= E\left[Y^2\right]/2y \\ &= \frac{y}{2} + \frac{Var[Y]}{2y} \end{aligned} \tag{14}$$

and the mean number of updates missed is

$$E[\text{updates missed}] = \frac{\lambda_u}{2}\left(y + \frac{Var[Y]}{y}\right). \tag{15}$$

The **obsolescence** seen by a query is the time interval between the latest snapshot and the last update after that, if any (see Fig. 2). Suppose a query arrives at time $T_i + C < T_{i+1}$, and the updates in this period occur at $T_i + U_1$, $T_i + U_2$, ..., $T_i + U_J$. Since the arrival process of up-dates is Poisson, a well known theorem (see [26, Theorem 2.3]) tells us that the joint distribution of U_1, U_2, ..., U_J is the same as that of a sequence of independent random variables $U_1', U_2', \ldots, U_J'$ distributed *uniformly* in (0, C), and then sorted. Thus the mean obsolescence is given by

$$\begin{aligned} E\left[\max\left(U_1, U_2, \ldots, U_J\right)\right] &= E\left[\max\left(U_1', U_2', \ldots, U_J'\right)\right] \\ &= E\left[C\left(1 - \frac{1}{J+1}\right)\right] \\ &= E[C] - E\left[\frac{C}{J+1}\right]. \end{aligned} \tag{16}$$

The current life C is distributed according to (11) and its mean value is given by (14).

$$\begin{aligned} E\left[\frac{C}{J+1}\right] &= E_C\left[E\left[\frac{C}{J+1}C\right]\right] \\ &= E_C\left[C\sum_j \frac{e^{-\lambda_u C}(\lambda_u C)^j}{(j+1)\cdot j!}\right] \\ &= E_C\left[\frac{1 - e^{-\lambda_u C}}{\lambda_u}\right]. \end{aligned}$$

Using (13),

$$E\left[e^{-\lambda_u C}\right] = \left(1 - E\left[e^{-\lambda_u Y}\right]\right)\Big/\lambda_u y. \tag{17}$$

Combining (14), (15), (16), (17) we have

$$E[\text{obsolescence}] = \frac{E[Y]}{2} + \frac{Var[Y]}{2E[Y]} - \frac{1}{\lambda_u} + \frac{1 - E\left[e^{-\lambda_u Y}\right]}{\lambda_u^2 E[Y]}. \tag{18}$$

The expectation $E[e^{-\lambda_u Y}]$ was computed in (10), and our derivation of the mean obsolescence is complete.

3.2 Some Asymptotic Results

When the arrival rate of queries λ_q becomes large compared to the service rate μ, $e^{-\rho_i}$ goes to zero, and the distribution (3) of intersnapshot times approaches an Extreme Value distribution, also known as a Gumbel distribution; see [25], [27].

$$Pr[Y < t] \to exp(-\rho_i e^{-\mu t}). \tag{19}$$

The corresponding mean and variance are

$$\begin{aligned} E[Y] &= \frac{1}{\mu}\left(\ln \rho_i + \gamma\right) \quad \text{where } \gamma \text{ is Euler's constant} \\ &= \frac{1}{\mu}\left(\ln\left(\frac{\lambda_q}{\mu}\right) + \ln\left(1 - e^{-\mu E[Y]}\right) + \gamma\right) \end{aligned} \tag{20}$$

$$Var[Y] = \frac{\pi^2}{6\mu^2}. \tag{21}$$

The difficult-looking (20) is really a quadratic equation in disguise:

$$\mu e^{-\gamma}(e^{\mu E[Y]})^2 - \lambda_q(e^{\mu E[Y]}) + \lambda_q = 0.$$

One of the two roots of this equation can be eliminated because it is a decreasing function of λ_q, which yields

$$E[Y] = \frac{1}{\mu}\ln\left(\frac{\lambda_q + \sqrt{\lambda_q^2 - 4\lambda_q \mu e^{-\gamma}}}{2\mu e^{-\gamma}}\right). \tag{22}$$

These asymptotic values are matched well with simulations (see Fig. 4). As (22) and (21) indicate, the mean inter-snapshot time $E[Y]$ increases without bound with λ_q; however, the variance is bounded for fixed μ. The probability distribution function of Y is thus sharply spiked around the mean when λ_q is large; this further justifies the approximation (1).

In order to compute the asymptotic values of the performance metrics, we need, other than the the mean and variance of Y given above, the following expectation:

$$E\left[e^{-\lambda_u Y}\right] \approx \rho^{-\lambda_u/\mu}\Gamma\left(1 + \lambda_u/\mu\right). \tag{23}$$

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

This follows from (10) by letting $\rho \to \infty$, or directly from the Laplace transform of the Extreme value distribution.

Substituting the expected values (20), (21), (22), (23) in (8), (15), and (18) for the mean number of versions, updates missed and query obsolescence respectively immediately yields the values of these measures in terms of the model parameters λ_q, λ_u, and μ. Comparison with simulation results (see Figs. 4 and 5) indicates that the asymptotic results are a good approximation when the rate of arrival of queries λ_q is high compared to the service rate μ (say, $\lambda_q > 4\mu$).

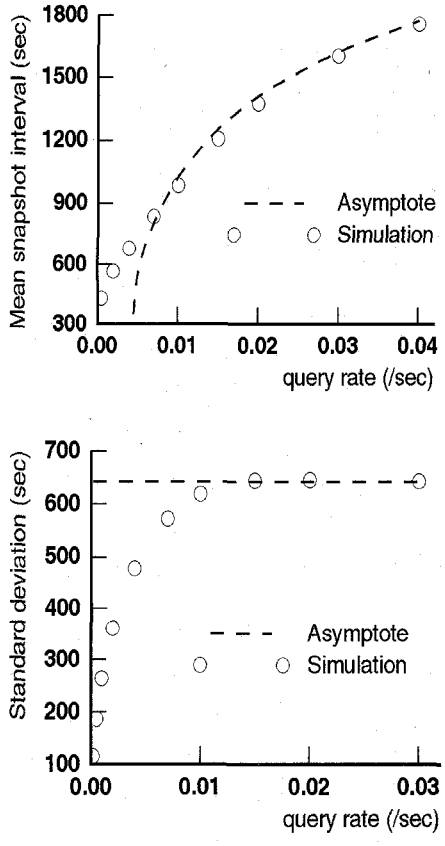


Fig. 4. Mean intersnapshot times for varying λ_q (Mean query length = 500 sec, Δ = 0.97* average snapshot interval).

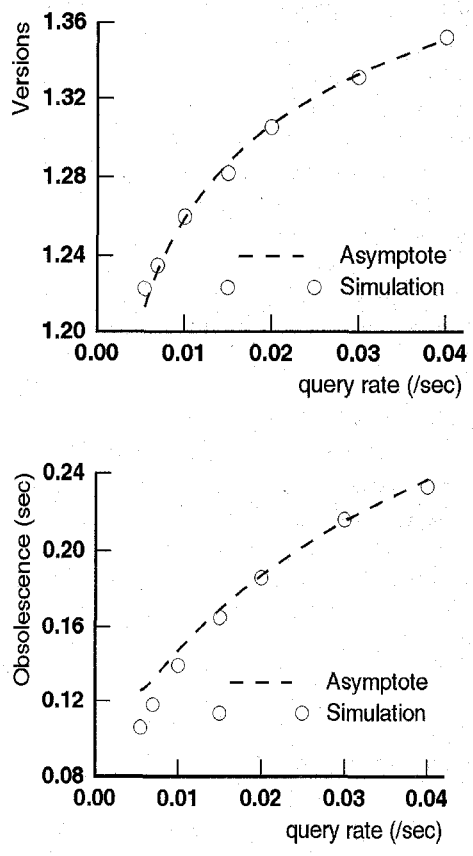


Fig. 5. Mean versions/page and obsolescence for varying λ_q (Mean query length = 500 sec, Δ = 0.97* average snapshot interval).

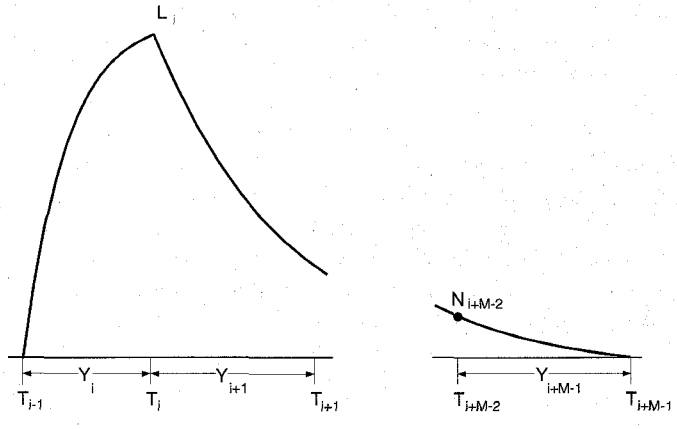


Fig. 6. Queries pending against Query snapshot taken at T_{i-1} (smoothed).

3.3 The Case of Multiple Snapshots

Our basic analytic approach in the case of $M > 2$ snapshots is conceptually similar to that in the case of two snapshots, although the details are different. The basic idea is to estimate the mean and variance of an intersnapshot interval Y_{i+M-1} in terms of the means and variances of the preceding $M - 1$ intersnapshot intervals $Y_i, \ldots, Y_{i+M-2}$. At steady state, the distributions of all the intersnapshot intervals are equal; hence, this yields equations for the stationary mean and variance of the intersnapshot interval. Once these moments are available, we can apply a renewal process approximation as in Section 3.1 to estimate the performance metrics.

To understand this better, refer to Fig. 6, which shows a smoothed sample path for the number of queries pending against the query snapshot taken at time T_{i-1}. This number increases until time T_i, when another snapshot is taken, and queries which arrive later are run against the newer snapshot. The number of queries at this time T_i is denoted L_i. The number of queries pending against this snapshot decreases as queries are completed. At time T_{i+M-2}, this snapshot becomes the oldest snapshot, and the number of queries pending against it is N_{i+M-2}. The length of time until the next snapshot is taken is Y_{i+M-1}, and it depends on N_{i+M-2}: if this number is zero, then the next snapshot is taken after time Δ_{i+M-1}, else when all the N_{i+M-2} queries complete.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

We proceed as follows to estimate the mean and variance of Y_{i+M-1}:

1) We estimate the mean and variance of L_i in terms of the mean and variance of Y_i.
2) We estimate the mean and variance of N_{i+M-2} in terms of the means and variances of $L_i, Y_{i+1}, \ldots, Y_{i+M-2}$.
3) We estimate the mean and variance of Y_{i+M-1} in terms of the mean and variance of N_{i+M-2}.
4) Assuming steady state, we set equal the corresponding moments of $Y_{i-1}, \ldots, Y_{i+M-1}$; this gives a set of equations which can be solved for these moments.

3.3.1 Estimating L_i

As we shall see later, to estimate L_i and N_{i+M-2} requires not only the mean and variance of Y_i, but the entire distribution. We therefore choose to approximate the stationary distribution of inter-snapshot times Y with a gamma distribution of the same mean and variance. A gamma random variable X with parameters $\lambda > 0$ and $\alpha > 0$ is a positive random variable with mean α/λ, variance α/λ^2, probability distribution function

$$f_X(x) = \frac{\lambda}{\Gamma(\alpha)}(\lambda x)^{\alpha-1}e^{-\lambda x} \quad \text{for } x > 0$$

which has the Laplace transform

$$E\left[e^{-sX}\right] = \int_0^\infty e^{-sx} f_X(x)dx = \left(\frac{\lambda}{\lambda+s}\right)^\alpha. \tag{24}$$

This distribution was chosen because it can approximate an extremely wide variety of distributions from exponential (by setting $\alpha = 1$) to normal (by taking α large) and is reasonably tractable analytically. Note that only the form of the distribution is approximated here; the moments will come from the analysis. Let

$$L_i \equiv P_{M-1}(T_i-)$$

denote the number of active queries accessing QS_{M-1} just before the ith snapshot advance. We assume that the system has achieved stationary state. As in the case $M = 2$, this can be modeled as a Birth-Death process with birth rates λ_q and death rates $\mu_j = j\mu$, and if the inter-snapshot interval was t, then L_i is a Poisson random variable with mean $\lambda_q(1-e^{-\mu t})/\mu$. Thus,

$$\begin{aligned} E[L_i] &= E\left[\frac{\lambda_q}{\mu}\left(1-e^{-\mu Y_i}\right)\right] \\ &= \frac{\lambda_q}{\mu}\left(1-E\left[e^{-\mu Y_i}\right]\right) \\ &= \frac{\lambda_q}{\mu}\left(1-\left(\frac{\lambda}{\lambda+\mu}\right)^\alpha\right) \end{aligned} \tag{25}$$

$$\begin{aligned} E\left[L_i^2\right] &= E\left[\frac{\lambda_q}{\mu}\left(1-e^{-\mu Y_i}\right)+\frac{\lambda_q^2}{\mu^2}\left(1-e^{-\mu Y_i}\right)^2\right] \\ &= \left(\frac{\lambda_q}{\mu}+\frac{\lambda_q^2}{\mu^2}\right)-\left(\frac{\lambda_q}{\mu}+\frac{2\lambda_q^2}{\mu^2}\right)\left(\frac{\lambda}{\lambda+\mu}\right)^\alpha+\frac{\lambda_q^2}{\mu^2}\left(\frac{\lambda}{\lambda+2\mu}\right)^\alpha \end{aligned} \tag{26}$$

and,

$$Var[L_i] = E\left[L_i^2\right] - E[L_i]^2. \tag{27}$$

3.3.2 Estimating N_{i+M-2}

The snapshot taken at time T_{i-1} becomes QS_0, the "oldest" snapshot at T_{i+M-2}, and the queries in it which survive until this time number N_{i+M-2}. Let

$$A_j \equiv \mathbf{1}(j\text{th query accessing } QS_{M-1} \text{ at time } T_i- \text{ survives until } T_{i+M-2}).$$

Clearly the As are identically distributed. Then

$$N_{i+M-2} = A_1 + A_2 + \cdots A_{L_i}$$

$$E[N_{i+M-2}] = E[A] \cdot E[L_i] \tag{28}$$

$$Var[N_{i+M-2}] = E[A]^2 Var[L_i] + E[L_i]Var[A]. \tag{29}$$

The mean and variance equations above are standard for random sums of identically distributed random variables; equation (28) is known as Wald's equation (see [26], [28]). Since the remaining service time of the queries is exponentially distributed,

$$\begin{aligned} E[A] &= E\left[exp\left(-\mu\left(Y_{i+1}+\cdots+Y_{i+M-2}\right)\right)\right] \\ &\approx E\left[exp(-\mu Y)\right]^{M-2} \\ &= \left(\frac{\lambda}{\lambda+\mu}\right)^{(M-2)\alpha} \end{aligned} \tag{30}$$

$$\begin{aligned} Var[A] &= E\left[A^2\right] - E[A]^2 = E[A] - E[A]^2 \quad \text{since } A^2 = A \\ &= \left(\frac{\lambda}{\lambda+\mu}\right)^{(M-2)\alpha} - \left(\frac{\lambda}{\lambda+\mu}\right)^{2(M-2)\alpha} \end{aligned} \tag{31}$$

Equations (25)-(31) give us the mean and variance of N_{i+M-2} in terms of λ and α. Since we have assumed the system to be in steady state at T_i, these give us the stationary mean and variance of N.

3.3.3 Estimating the Moments of Y_{i+M-1}

For the next step, which is to estimate the mean and variance of Y_{i+M-1}, we find that we need not just the mean and variance, but the entire distribution of N_{i+M-2}. We choose to approximate the distribution of N_{i+M-2}. by a Negative Binomial (Pascal) distribution with the parameters $\beta > 0$, $0 < p < 1$ and $q \equiv 1 - p$. A random variable N so distributed has the mean $\beta q/p$, variance $\beta q/p^2$, probability mass function

$$Pr[N = n] = \binom{\beta+n-1}{n} p^\beta q^n \quad \text{for } n = 0, 1, \ldots$$

and probability generating function

$$E\left[z^N\right] = \left(\frac{p}{1-qz}\right)^\beta.$$

A Negative Binomial distribution is the discrete analog of a gamma distribution, and approximates a large variety of discrete distributions from Geometric (by setting $\beta = 1$) to

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

Poisson (by taking β large). The values of p, q, and β are chosen so that the mean and variance of this distribution match the mean and variance of N_{i+M-2} previously computed:

$$p = \frac{E[N_{i+M-2}]}{Var[N_{i+M-2}]}; \quad q = 1 - p; \quad \beta = \frac{E[N_{i+M-2}]p}{q}. \tag{32}$$

Now proceeding as in (2), we find

$$Pr[Y_{i+M-1} < t] = E\left[\left(1 - e^{-\mu t}\right)^{N_{i+M-2}}\right] - Pr[N_{i+M-2} = 0]Pr[\Delta_{i+M-1} > 0]$$
$$= \left(\frac{p}{p + qe^{-\mu t}}\right)^{\beta} - p^{\beta} Pr[\Delta_{i+M-1} > 0].$$

Therefore,

$$E[Y_{i+M-1}] = \int_0^{\infty} Pr[Y_{i+M-1} > t]dt$$
$$= \int_0^{\infty} \left(1 - \left(\frac{p}{p + qe^{-\mu t}}\right)^{\beta}\right)dt + p^{\beta} E[\Delta_{i+M-1}].$$

$$E[Y_{i+M-1}^2] = \int_0^{\infty} 2tPr[Y_{i+M-1} > t]dt$$
$$= \int_0^{\infty} 2t\left(1 - \left(\frac{p}{p + qe^{-\mu t}}\right)^{\beta}\right)dt + p^{\beta} E[\Delta_{i+M-1}^2].$$

Again we are faced with integrals without a simple closed form; however they are easy to evaluate numerically. We may then determine α and λ in terms of these:

$$\lambda = \frac{E[Y_{i+M-1}]}{Var[Y_{i+M-1}]}; \quad \alpha = \frac{E[Y_{i+M-1}]^2}{Var[Y_{i+M-1}]}. \tag{33}$$

Equations (32) and (33) give us a system of simultaneous equations in p, q, β, α, and λ. While there is no simple solution for these that is apparent, they can usually be solved quite rapidly using a fixed-point method. (In most cases where we chose Δ_i with a fixed distribution, the method converged to a 0.1% accuracy within five iterations.)

3.3.4 Calculating the Performance Measures

Once the values of α and λ have been found it is easy to determine the mean values of our performance measures. As in the last section, it is assumed that Y_0, Y_1, ... are *i.i.d.* random variables, in this case distributed according to the gamma distribution with parameters λ and α.

The **number of versions** stored per page is given by (6) of the the last section. The number of copies at time $t \in (T_i, T_{i+1})$ is

$$NCopies(t) = 1 + \sum_{1}^{M-1} \mathbf{1}\left(\text{update in } \left(T_{i-j}, T_{i-j+1}\right)\right) + \mathbf{1}\left(\text{update in } \left(T_i, t\right)\right).$$

$$E\left[\int_{T_i}^{T_{i+1}} NCopies(t)dt\right]$$
$$= E\left[Y_{i+1} + \sum_{j=1}^{M-1} Y_{i-j+1}\left(1 - e^{-\lambda_u Y_{i-j}}\right) + \int_{T_i}^{T_{i+1}} \left(1 - e^{-\lambda_u (t - T_i)}\right)dt\right].$$

Substituting in (6) and simplifying, the mean number of versions stored is

$$\lim_{t\to\infty} \frac{1}{t} \int_0^t NCopies(x)dx$$
$$= M + 1 - \frac{1}{\lambda_u E[Y]} - \left(M - 1 - \frac{1}{\lambda_u E[Y]}\right) E\left[e^{-\lambda_u Y}\right]$$
$$= M + 1 - \frac{\lambda}{\alpha\lambda_u} - \left(M - 1 - \frac{\lambda}{\alpha\lambda_u}\right)\left(\frac{\lambda}{\lambda + \lambda_u}\right)^{\alpha}. \tag{34}$$

The mean **number of updates missed** is given by (15):

$$E[\text{missed updates}] = \frac{\lambda_u}{2}\left(E[Y] + \frac{Var[Y]}{E[Y]}\right)$$
$$= \frac{\lambda_u(\alpha + 1)}{2\lambda}. \tag{35}$$

The mean **obsolescence** is given by (18):

$$E[\text{obsolescence}] = \frac{E[Y]}{2} + \frac{Var[Y]}{2E[Y]} - \frac{1}{\lambda_u} + \frac{1 - E\left[e^{-\lambda_u Y}\right]}{\lambda_u^2 E[Y]}$$
$$= \frac{1 + \alpha}{2\lambda} - \frac{1}{\lambda_u} + \frac{\lambda}{\alpha\lambda_u^2}\left(1 - \left(\frac{\lambda}{\lambda + \lambda_u}\right)^{\alpha}\right). \tag{36}$$

4 Validation and Performance Analysis

In this section, we present a validation of the approximations made in the solutions of our analytical models by comparing the analytical results with simulation data. We analyze the performance of DFV with two or more snapshots by examining the trade-offs between storage cost and obsolescence. Finally, we explore other completion-triggered and arrival-triggered QS-advance policies.

4.1 DFV with Two QSs

This section presents the performance analyses of DFV using only two QSs. In both analysis and simulation, we assumed that the QS-advance strategy is the base case; i.e., completion-triggered, only QL_0 empty. The Δ chosen for the system-scheduled QS-advance is 0.97 × the average snapshot interval. The reason for choosing 0.97 will be explained in Section 4.2. Additionally, we assumed that queries access the entire database uniformly, but transactions access the database following a $\phi - \psi$ skew rule: A fraction ϕ of transaction updates goes to a fraction ψ of the database; i.e., most of the updates go to the *hot* set which contains a small portion of the database, while only a small fraction of the updates go to the *cold* set containing the majority of the database. Within each set, the access is assumed to be uniformly distributed. The total database size is 100K pages; each transaction accesses 10 pages, and each access has a 20% chance of being an update. The effective update rate to a page in each set of data was calculated; separate analyses and simulations were carried out for a page in each set. The weighted average of these two analyses (simulations) gives the average storage overhead per page for the overall database, the mean query obsolescence and the mean number of updates missed by a query. The obsolescence is normalized by dividing it with the mean query length. We will show the storage overhead and query obsolescence for the hot set, the cold set and the overall database separately in some figures.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

In the entire study, we examined many parameter values in both analysis and simulation. The accuracy of the analytical models and the trend of performance trade-offs held true for all the parameter values studied. However, in this paper, unless otherwise mentioned, we chose the defaults for query arrival rate, transaction arrival rate to the system, mean query length, and $\phi - \psi$ to be 0.2, 50, 500, and 90% – 5%, respectively. This yields an update rate to each hot-set page of 0.018 updates/sec. Note that the default query arrival rate is much smaller than the default transaction arrival rate since ad hoc queries in general require a large number of reads and are much longer than update transactions.

Since the average snapshot interval is the basis for our analysis, we first show the effect of query arrival rate on the average snapshot interval. Fig. 7 shows the average snapshot interval for two different mean query lengths: 500 and 1,000 seconds. The average snapshot interval increases with query arrival rate and mean query length. Both simulation and analysis results are presented. The analysis matches very closely with simulation. The simulations in this paper were controlled by the accuracy of the average snapshot interval; for each simulation, the width of the 95% confidence interval of the average snapshot interval is ≤ 1% of the point estimate.

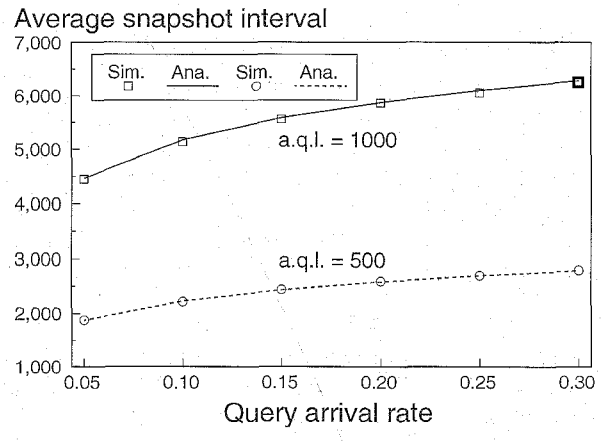


Fig. 7. Effect of query arrival rate on average snapshot interval (a.q.l. = average query length).

Figs. 8 and 9 show the effect of query arrival rate on the average number of versions per page. Notice that the actual storage overhead per page due to DFV is the average number of versions minus one. In Fig. 8, the weighted averages for the overall database are shown for three different transaction arrival rates: 10, 30, and 50 transactions/sec. In Fig. 9, the average for the hot set and the cold set are displayed for the case of 50 transactions/sec. Both simulation and analysis results are shown in these two figures, and they match closely. When the transaction update rates are high, as indicated by the curve for the hot set in Fig. 9, the storage overhead can be quite substantial, close to the worst case scenario where three copies are maintained for each page. For cases with skewed access or low transaction update rates, the storage cost is low to moderate and increases with transaction arrival rate, but it saturates quickly with the increase in query

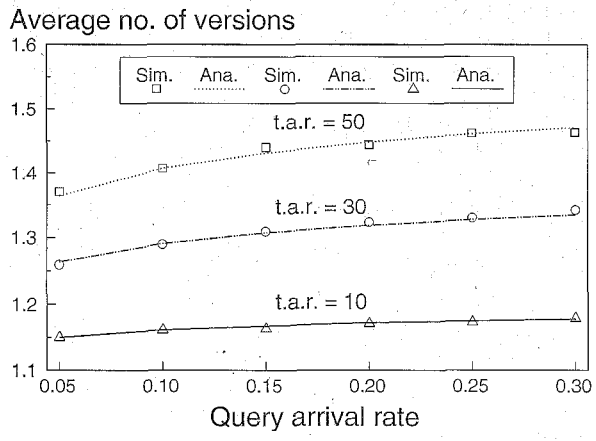


Fig. 8. Effect of query arrival rate on the average number of versions (t.a.r. = transaction arrival rate).

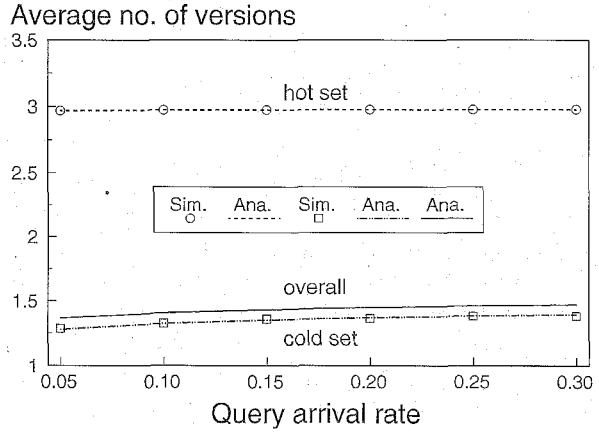


Fig. 9. Average number of versions for the hot set and the cold set, when t.a.r. = 50.

arrival rate. Generally, the longer the snapshot time, the more likely it is that a newly committed version will be created before a snapshot, resulting in higher storage overhead.

Figs. 10 and 11 show the obsolescence, and Fig. 12 illustrates the number of updates missed by a query. Again, the analysis results match almost exactly with simulation. As with the storage overhead, the obsolescence and the number of updates missed increase with the transaction update rate. Since these two measures are quite similar, we show only the obsolescence in the remainder of the paper.

4.2 Trade-Offs Among QS-Advance Strategies

This section shows the design trade-offs among different strategies for advancing query snapshots when the number of QSs used is larger than 2. Before going into the trade-offs, we validate the analytical models by demonstrating both analysis and simulation results on the average snapshot interval for two different query arrival rates λ_qs. As Fig. 13 indicates, the analytical model for more than two QSs also matches accurately with simulation. As we expect, when M increases, the average snapshot interval decreases.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

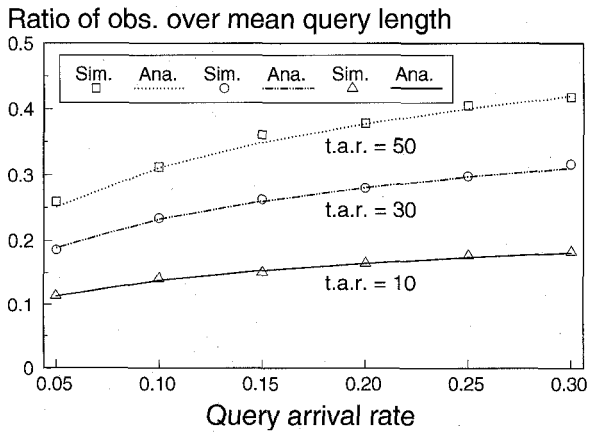


Fig. 10. Effect of query arrival rate on the ratio of obsolescence over mean query length.

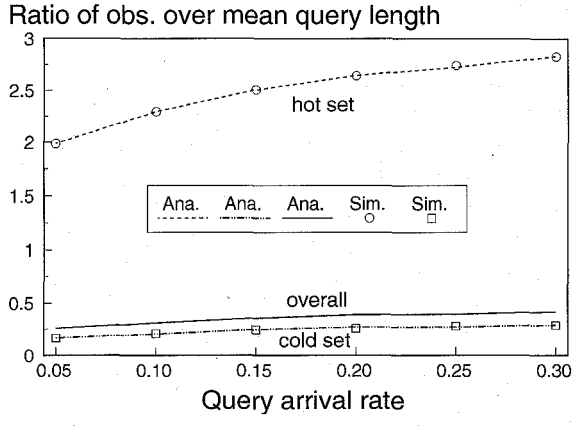


Fig. 11. Ratio of obsolescence over mean query length for the hot set and cold set, when t.a.r. = 50.

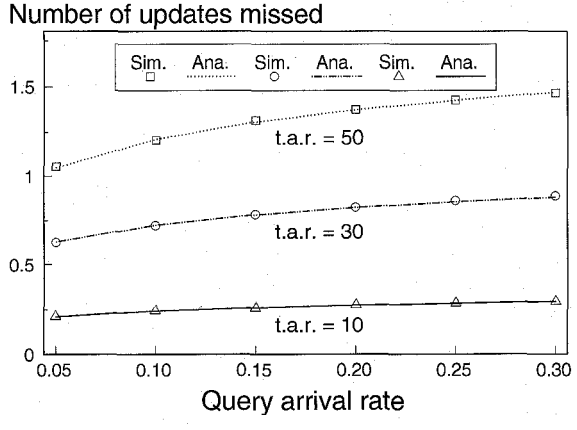


Fig. 12. Effect of query arrival rate on the number of updates missed.

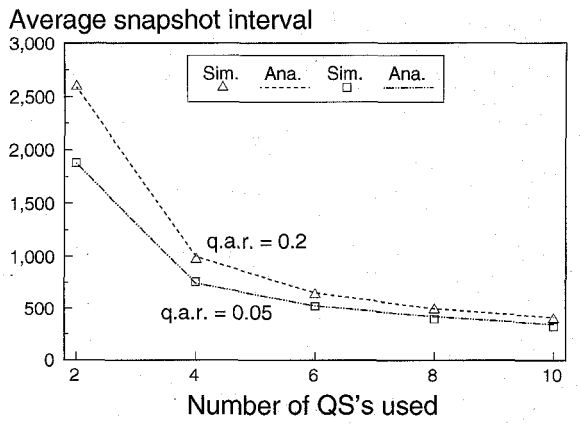


Fig. 13. Average snapshot interval for different λ_q (q.a.r. = query arrival rate, λ_q).

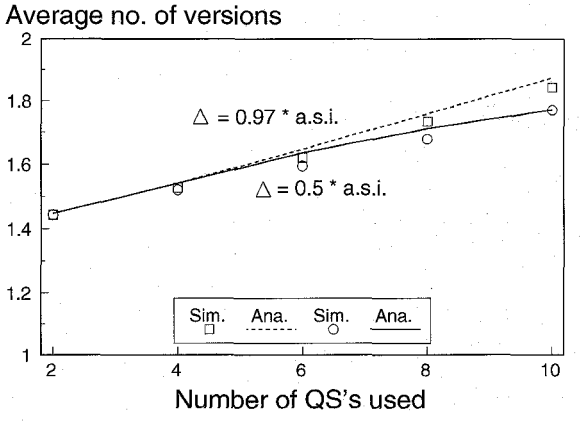


Fig. 14. Effect of Δ on the storage overhead (a.s.i. = average snapshot interval).

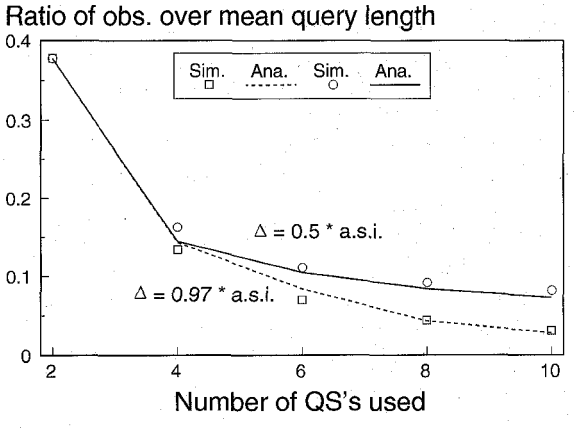


Fig. 15. Effect of Δ on query obsolescence (a.s.i. = average snapshot interval).

4.2.1 The Effect of Δ

We first show the effect of different choices of Δ on the storage overhead and query obsolescence. Figs. 14 and 15 show the average storage cost and query obsolescence, respectively, for different Ms using the base strategy to schedule the QS-advance. We assumed that $\Delta = k \times$ the average snapshot interval, and examined two different ks: 0.97 and 0.50. In the simulation, we chose each Δ_i according to an exponential distribution with mean equal to the average query length divided by $M - 1$ for the initial 100 snapshot advances. After that, each Δ_i was chosen to be the average of all the previous snapshot intervals. Both simulation and analysis results are shown in these two figures, and they too match closely. From these two figures, we observe the general trend of the trade-off between storage cost and query obsolescence, and the

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

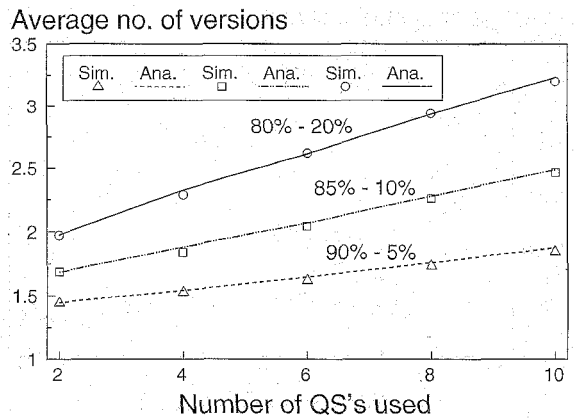


Fig. 16. Impact of skew on storage overhead.

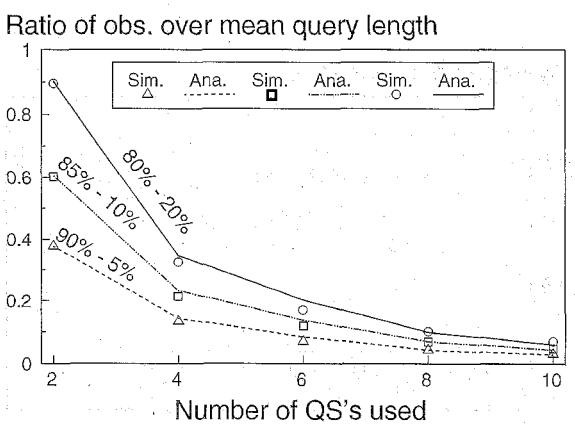


Fig. 17. Impact of skew on query obsolescence.

sensitivity of both to different Δs, as M becomes large. Also illustrated in Figs. 14 and 15 is the fact that when M increases beyond a certain number, the rate of improvement in query obsolescence diminishes even though the storage cost increases at almost the same rate. We also examined the effect of the distribution of Δ. Two cases were considered: constant and with an exponential distribution, and the results showed that the difference is negligible.

In the study, if $k \geq 1$, we found that the simulation may not converge for small query arrival rates, λ_q. For moderate to large λ_q, however, k can be ≥ 1. This is because when λ_q is small, QL_1 is likely to be empty before a QS-advance; therefore, Δs become dominant in the computation of the average snapshot interval. Thus, if λ_q is very small and we choose $k = 1$, then the average snapshot length is approximately determined by Δ_1, the initial value for Δ, and if we choose $k > 1$, then the average snapshot length may increase without bound. In this paper, we chose a k that is < 1. Trial and error showed that $k \approx 0.97$ yields the lowest obsolescence, and hence this value was used in most cases.

4.2.2 The Effect of Skew

To further validate the analytical models developed in Section 3, we show both analysis and simulation results on the effect of skew. (For the method used to obtain the analysis and simulation results for the skewed cases, please see Section 4.1.) Figs. 16 and 17 show three different cases of skew. In general, the more skewed the access, the better the performance is in average storage overhead and query obsolescence.

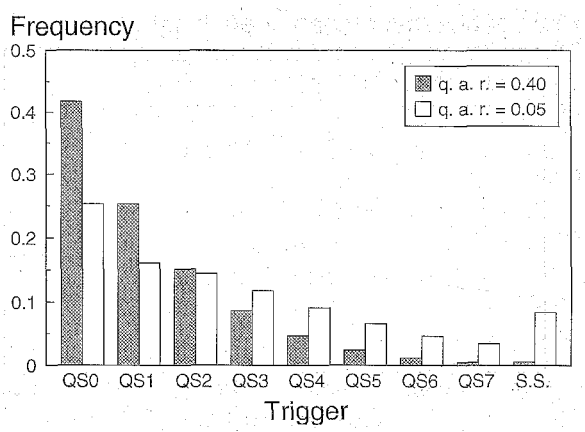


Fig. 18. Frequency of QS-advance by each QL_i's becoming empty (S.S. = system scheduled).

4.2.3 The Effect of Blocking

The completion-triggered strategy discussed so far has a *blocking effect*: when a QL_i, where $i \neq 0$, becomes empty, it has to wait i QS-advances before it can be discarded. If this blocking effect is removed, both storage overhead and query obsolescence can be improved. Using simulation, we studied the effect of removing this blocking; i.e., we studied the relative frequency with which the snapshot is triggered by a QL_i becoming empty. Two different query arrival rates, $\lambda_q = 0.4$ and 0.05, are shown in Fig. 18. From the shape of the bar charts in Fig. 18, this blocking effect can have a significant impact on the system performance, especially if the query arrival rate is low. In other words, if allowed, a substantial number of snapshots can be advanced much earlier, resulting in a reduction in both average storage cost and query obsolescence.

Another approach which can further improve the query obsolescence is to use the arrival-triggered strategy; that is, a QS-advance is triggered upon a query arrival if there is some empty QL_i. A generalization of the arrival-triggered approach is to have at least K queries access each QS; that is, a snapshot advance is triggered only when there have been at least K queries accessing QS_{M-1} and there is some empty QL_i upon a query arrival. We examined three different values of K: 1, 2, and 3, and found that there is no difference between them when the query arrival rate λ_q is 0.2 and only a small difference when λ_q is 0.05. As a result, in the following studies, we chose $K = 1$.

Figs. 19 and 20 show the impacts of different strategies on the average storage cost and query obsolescence for the case with skewed access, Figs. 21 and 22 for the hot set, and Figs. 23 and 24 for the cold set, respectively. Three different strategies were compared: completion-triggered with and without blocking, and arrival-triggered. From these figures, it is seen that by removing the blocking effect, both the average storage overhead and query obsolescence are improved. In addition, there is little difference between the completion-triggered strategy with the blocking effect removed and the arrival-triggered strategy. This is so, in part, because the default query arrival rate λ_q (0.2) is high enough such that there is always a query arrival soon after any QL_i becomes empty.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

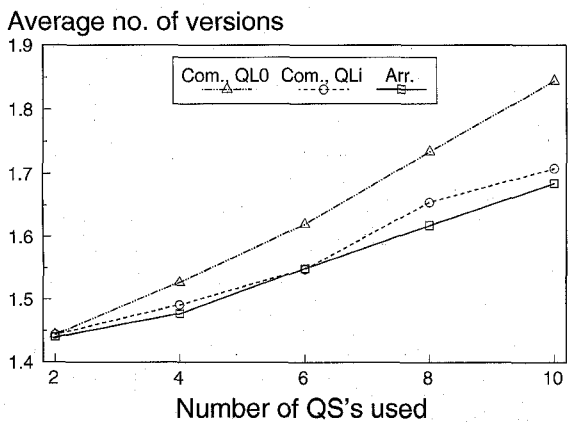


Fig. 19. Effect of QS-advance strategies on storage overhead for the overall database (QL0: with blocking, QLi: without blocking).

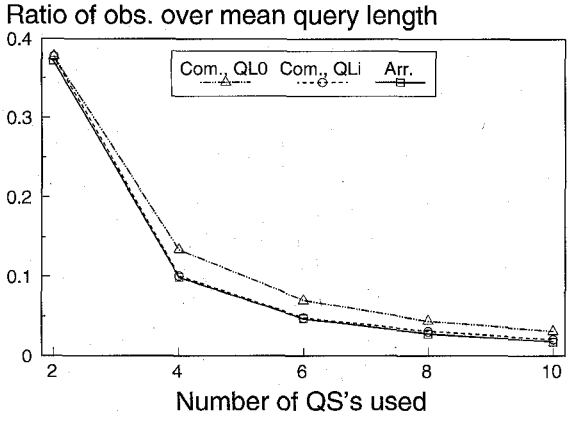


Fig. 20. Effect of QS-advancing strategies on query obsolescence for the overall database.

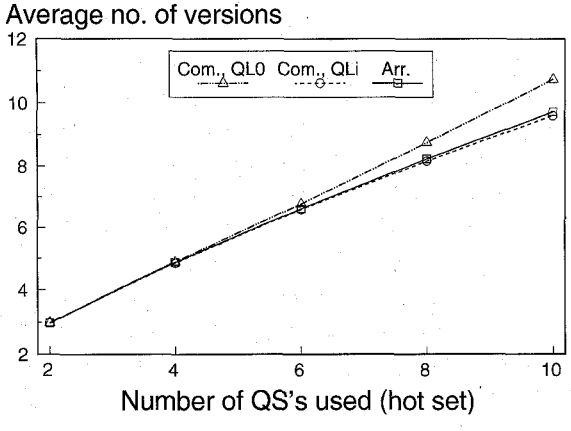


Fig. 21. Effect of QS-advancing strategies on storage overhead for the hot set.

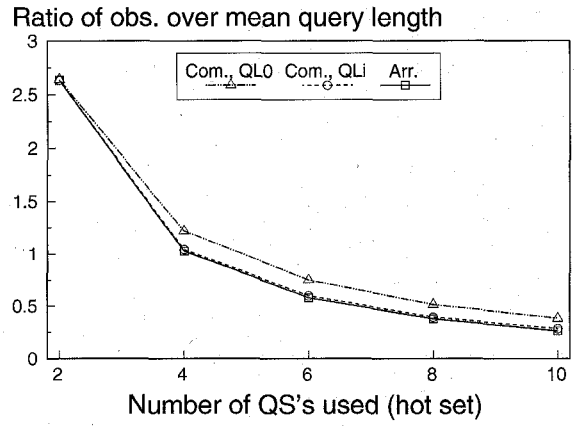


Fig. 22. Effect of QS-advancing strategies on query obsolescence for the hot set.

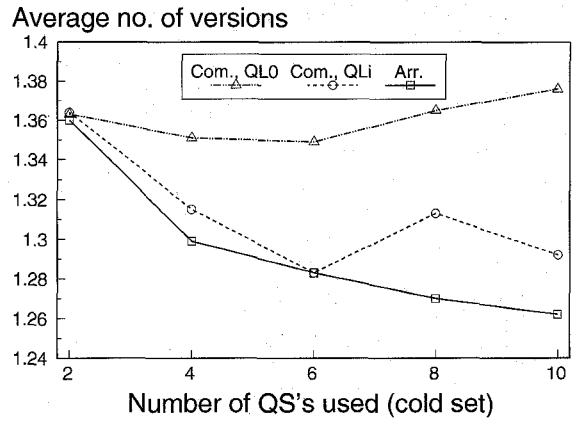


Fig. 23. Effect of QS-advancing strategies on storage overhead for the cold set.

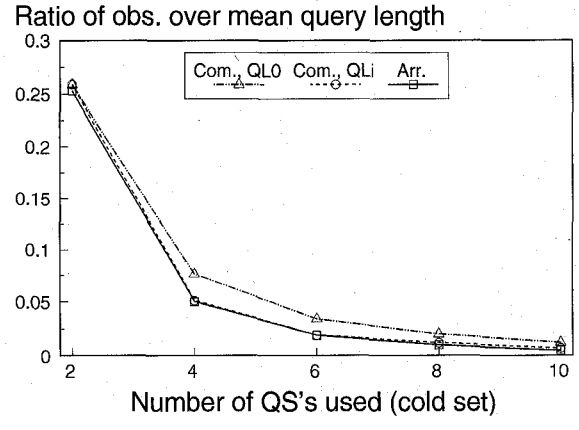


Fig. 24. Effect of QS-advancing strategies on query obsolescence for the cold set.

For the cases with high transaction update rates (as in the hot set), as demonstrated in Figs. 21 and 22, even though the obsolescence is generally decreased when M increases, such a reduction is at the cost of a substantial increase in the amount of storage cost. However, for the cases with skewed access and low transaction update rates, as demonstrated in Figs. 19, 20, 23, and 24, with a slight increase in the number of QSs beyond 2, query obsolescence can be substantially reduced while the storage overhead increases only slightly or even decreases as in Fig. 23. When transaction update rates are very low (as in the cold set), as shown in Figs. 23 and 24, the query obsolescence can be reduced to almost zero without increasing the storage overhead.

In Fig. 23, when transactrion update rates are very small, the storage overhead decreases as the number of QSs increases. This is due to the fact that with very low update rates, there is, with high probability, at most one update to a page within a snapshot interval. As the number of QSs in-

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

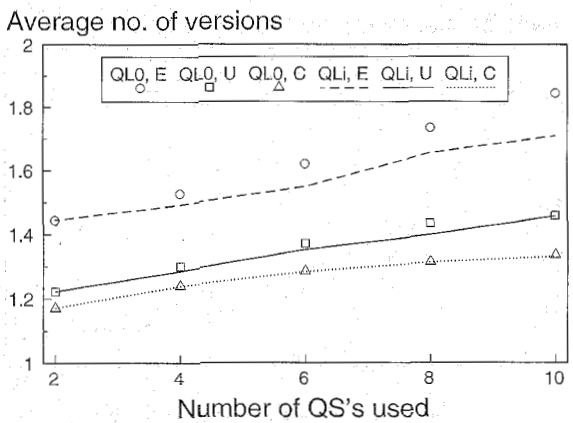


Fig. 25. Effect of query length distribution on storage cost for the overall database (E: exponential, U: uniform, C: constant).

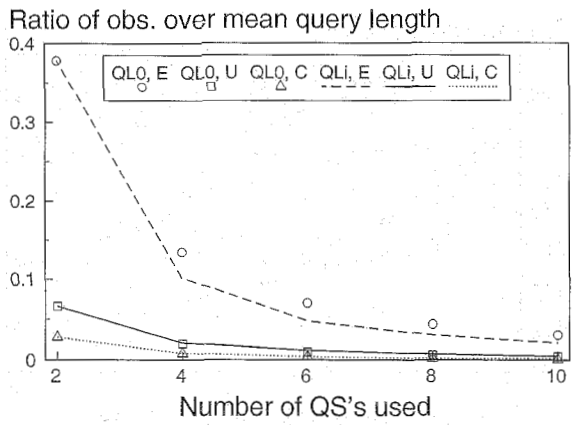


Fig. 26. Effect of query length distribution on query obsolescence for the overall database.

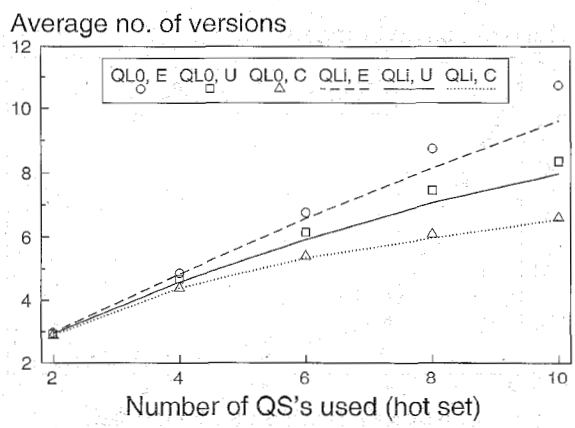


Fig. 27. Effect of query length distribution on storage cost for the hot set.

creases, the average snapshot length decreases, thus decreasing the average length of time that a given version of the page exists. (The possibility of a decrease in the storage overhead with an increase in the number of QSs was independently shown in [29].)

4.2.4 The Effect of Query Length Distribution

So far, we have studied the performance of DFV assuming that the distribution of query execution times is exponential, whose coefficient of variation $C_X = 1$. (The coefficient of variation of a random variable X, denoted by C_X, is defined as $\sigma_X/\bar{x}$, where σ_X is the standard deviation and $\bar{x}$ is the mean.) Here we also examine the effect of blocking and the trade-offs between storage cost and obsolescence of DFV using other query length distributions with the same mean but different coefficients of variation.

First, we studied the effects of query length distributions with $C_X < 1$. Figs. 25 and 26 show the average storage cost and obsolescence for the overall database (with skewed access), Figs. 27 and 28 for the hot set, and Figs. 29 and 30 for the cold set, respectively. Two distributions with $C_X < 1$, along with the exponential one, are shown in these figures: a uniform distribution with $C_X = 1/\sqrt{3}$ and a constant distribution with $C_X = 0$. In the case of uniform distribution, the query length is uniformly distributed between 0 and $2/\mu$ seconds, where $1/\mu$ is the mean query execution time used in the exponential distribution. In the case of constant distribution, every query is serviced for $1/\mu$ seconds. As the coefficient of variation decreases from 1 to 0, the storage cost and obsolescence decrease (see Figs. 25 and 26); the amount of reduction in obsolescence also decreases when the number of snapshots is increased from two to four (see Figs. 26, 28, and 30); and the significance of blocking effect diminishes as well (see Figs. 25, 27, and 29). However, the general trade-offs between storage cost and obsolescence observed in Section 4.2.3 still hold true as the number of QSs increases (compare Figs. 27–30 with Figs. 21–24). In other words, for low update rates, we can still use more QSs to lower obsolescence without increasing the storage cost for the cases of query length distributions with CX < 1 (see Figs. 29 and 30).

We also examined the effects of query length distributions with $C_X > 1$. Figs. 31 and 32 show the average storage cost and obsolescence for the overall database, Figs. 33 and 34 for the hot set, and Figs. 35 and 36 for the cold set, respectively. Two hyperexponential distributions with parameters $(1/5, \mu/2, 4\mu/3)$ and $(1/4, 2\mu/5, 2\mu)$,[2] respectively, were studied, where $1/\mu$ is the mean query execution time used in the exponential distribution. The parameters for the two hyperexponential distributions were chosen such that their means are equal to $1/\mu$ and their C_Xs are different, $\sqrt{1.5}$ and $\sqrt{2.5}$, respectively. From these figures, as the coefficient of variation increases from 1 to $\sqrt{2.5}$, the storage cost and obsolescence increase steadily (see Figs. 31 and 32); the amount of reduction in obsolescence rises when the number of QSs is increased from 2 to 4 (see Figs. 32, 24, and 36); and the significance of blocking effect grows as well (compare Fig. 31 with Fig. 25). Thus, it is more effective to reduce obsolescence with little increase in storage cost by using more QSs when C_X is larger (see Figs. 35 and 36). As

2. A hyperexponential random variable X with parameters (p, μ_1, μ_2) is defined as $X = X_1$ with probability p and $X = X_2$ with probability $1 - p$, where X_1 and X_2 are statistically independent exponential random variables with rates μ_1 and μ_2, respectively, where $\mu_1 \neq \mu_2$, and X has mean $\bar{x} = p/\mu_1 + (1-p)/\mu_2$ and variance $\sigma^2 = 2(p/\mu_1^2 + (1-p)/\mu_2^2) - \bar{x}^2$ (see [22, p. 142]).

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

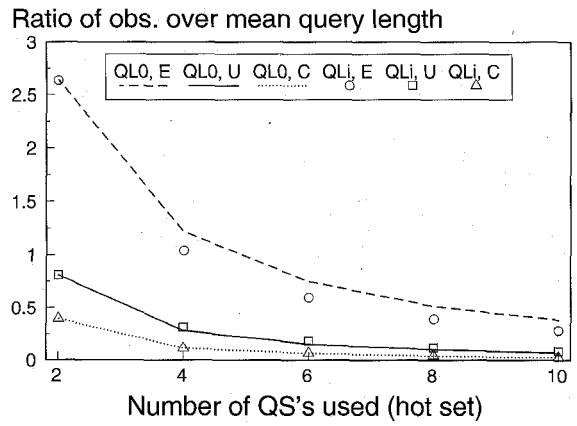


Fig. 28. Effect of query length distribution on query obsolescence for the hot set.

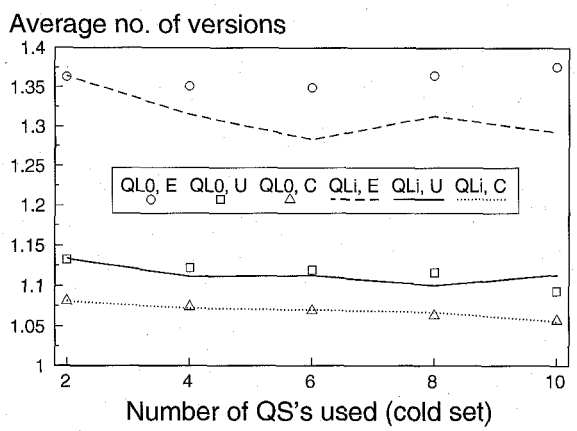


Fig. 29. Effect of query length distribution on storage cost for the cold set.

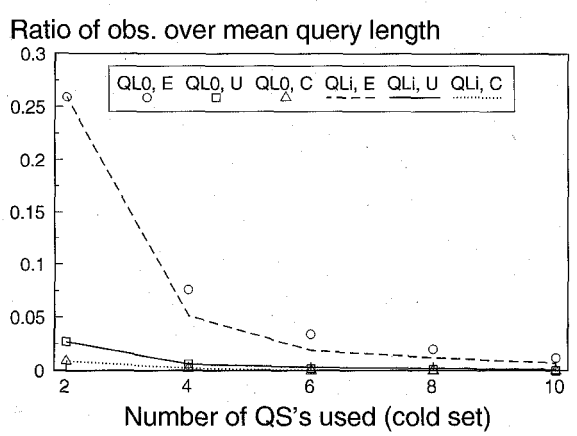


Fig. 30. Effect of query length distribution on query obsolescence for the cold set.

in the case of $C_X < 1$, the general trade-offs between obsolescence and storage cost observed in Section 4.2.3 also hold true for query length distributions with $C_X > 1$ (compare Figs. 33–36 with Figs. 21–24).

5 Conclusions

In this paper, we first developed analytical models for DFV using only two snapshots. A one-moment approximation was derived for the length of an inter-snapshot interval. The process of snapshot advances was then modeled approximately as a renewal process, and formulae were derived for the mean storage cost, mean obsolescence and the mean number of updates missed by a query. Closed-form asymptotic results were also given for these performance measures when the query arrival rates are high. Then, to generalize to more than two snapshots, we used a two-moment approximation for the inter-snapshot lengths and derived a set of recurrence relations for the two moments. A fixed point approach was used to deduce the values of the moments, and the query snapshot advance process was again treated as a renewal process to derive

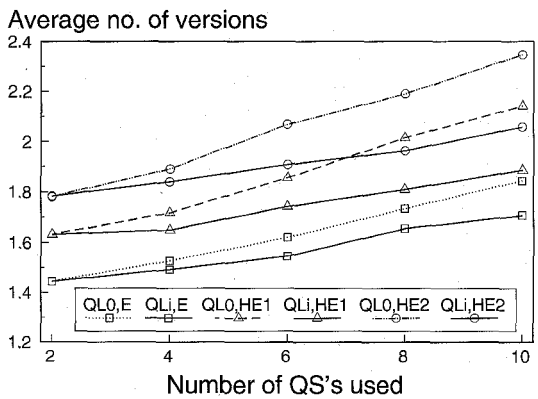


Fig. 31. Storage cost for the overall database (E: exponential, HE1: hyperexponential with $C_X^2 = 1.5$, HE2: hyperexponential with $C_X^2 = 2.5$).

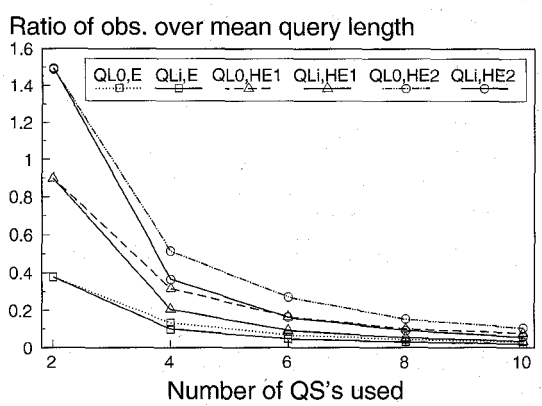


Fig. 32. Query obsolescence for the overall database.

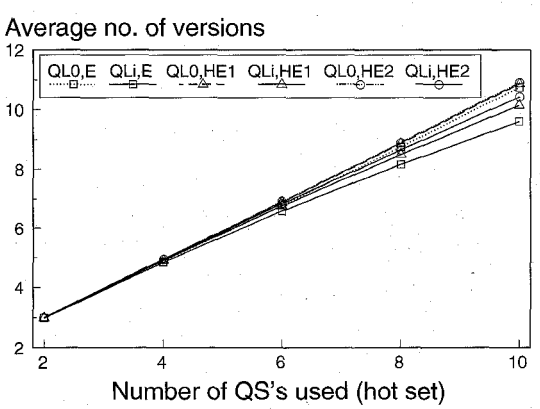


Fig. 33. Storage cost for the hot set.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

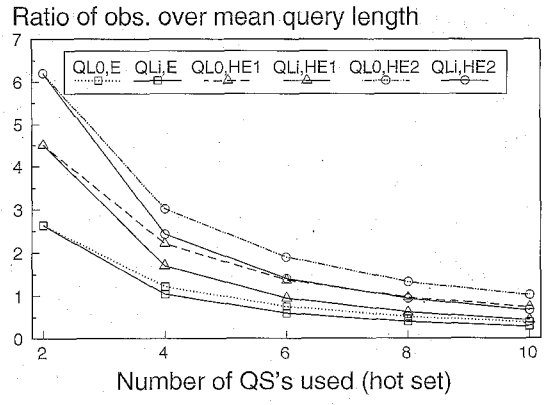


Fig. 34. Query obsolescence for the hot set.

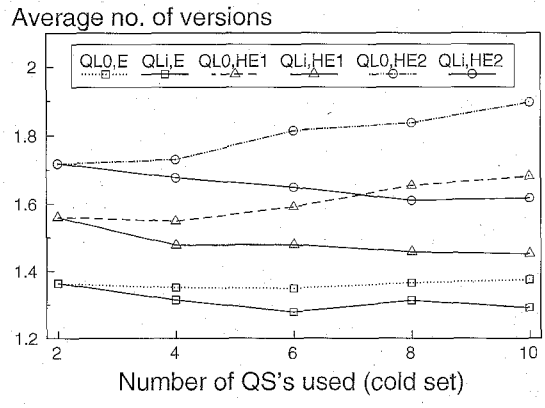


Fig. 35. Storage cost for the cold set.

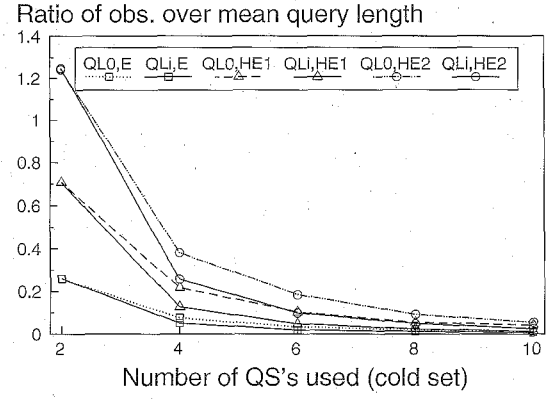


Fig. 36. Query obsolescence for the cold set.

the values of the performance metrics. Simulation was used to validate the analytical results and to examine the trade-off among different strategies for advancing query snapshots when more than two snapshots are used. Different completion-triggered, with and without blocking effect, and arrival-triggered strategies were studied.

The results show that

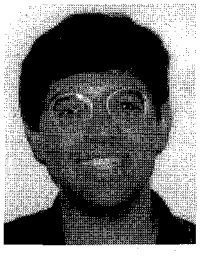
1) the analytical models match closely with simulation. The accuracy of the approximations made in the solutions of the analytical models was extensively validated by simulation.
2) Both the storage overhead and obsolescence are sensitive to the QS-advancing strategies, especially for $M > 2$ QSs. By removing the blocking effect, both the average storage overhead and query obsolescence are reduced. We also showed that the mean value of Δ, the time interval for system-scheduled snapshot advances, has an impact on the system performance, while the distribution of Δ does not.
3) Generally speaking, increasing the number of QSs demonstrates a trade-off between storage overhead and query obsolescence. For high update rates, even though increasing the number of QSs beyond 2 reduces the obsolescence, the storage overhead can increase substantially. However, for cases with skewed access or low update rates, a moderate increase in the number of QSs beyond 2 can substantially reduce the obsolescence, while the storage overhead may increase only slightly, or even decrease in some cases. For very low update rates, a large number of QSs can be used to reduce the obsolescence to almost zero without increasing the storage overhead. These observations hold true regardless of the query length distribution, i.e., whether its coefficient of variation is smaller than, equal to or larger than 1.
4) As the coefficient of variation of the query length distribution increases, the storage cost, the query obsolescence, and the effect of blocking all increase as well. However, increasing the number of snapshots reduces obsolescence more sharply, while the storage costs increase only slightly.

REFERENCES

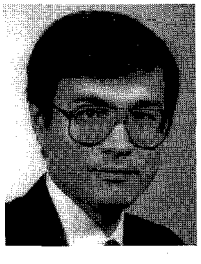
[1] H. Pirahesh et al., "Parallelism in Relational Data Base Systems: Architectural Issues and Design Approaches," *Proc. Second Int'l Symp. Databases in Parallel and Distributed Systems*, pp. 4–29, 1990.
[2] C. Pu, C.H. Hong, and J.M. Wha, "Performance Evaluation of Global Reading of Entire Databases," *Proc. First Int'l Symp. Databases in Parallel and Distributed Systems*, pp. 167–176, 1988.
[3] C. Pu, "On-the-Fly, Incremental, Consistent Reading of Entire Databases," *Algorithmica*, vol. 1, no. 3, pp. 271–287, Oct. 1986.
[4] M.J. Carey and W.A. Muhanna, "The Performance of Multiversion Concurrency Control Algorithms," *ACM Trans. Computer Systems*, vol. 4, no. 4, pp. 338–378, Nov. 1986.
[5] D.J. Haderle and R.D. Jackson, "IBM Database 2 Overview," *IBM Systems J.*, vol. 23, no. 2, pp. 112–125, 1984.
[6] K.-L. Wu, P.S. Yu, and C. Pu, "Divergence Control for Epsilon-Serializability," *Proc. Int'l Conf. Data Eng.*, pp. 506–515, 1992.
[7] C. Pu and A. Leff, "Replica Control in Distributed Systems: An Asynchronous Approach," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 377–386, 1991.
[8] P.A. Bernstein and N. Goodman, "Multiversion Concurrency Control—Theory and Algorithms," *ACM Trans. Database Systems*, vol. 8, no. 4, pp. 465–483, Dec. 1983.
[9] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
[10] R. Bayer, H. Heller, and A. Reiser, "Parallelism and Recovery in Database Systems," *ACM Trans. Database Systems*, vol. 5, no. 2, pp. 139–156, June 1980.
[11] D. Agrawal and S. Sengupta, "Modular Synchronization in Multiversion Databases: Version Control and Concurrency Control," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 408–417, 1989.
[12] A. Chan et al., "The Implementation of an Integrated Concurrency Control and Recovery Scheme," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 184–191, 1982.
[13] A. Chan and R. Gray, "Implementing Distributed Read-Only Transactions," *IEEE Trans. Software Eng.*, vol. 11, no. 2, pp. 205–212, Feb. 1985.
[14] D.P. Reed, "Implementing Atomic Actions on Decentralized Data," *ACM Trans. Computer Systems*, vol. 1, no. 1, pp. 3–23, Feb. 1983.
[15] W.E. Weihl, "Distributed Version Management for Read-Only Actions," *IEEE Trans. Software Eng.*, vol. 13, no. 1, pp. 55–64, Jan. 1987.
[16] P.M. Bober and M.J. Carey, "On Mixing Queries and Transactions via Multiversion LOCKING," *Proc. Int'l Conf. Data Eng.*, pp. 535–545, 1992.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.

[17] D.M. Dias, A. Goyal, and F.N. Parr, "An Intelligent Page Store for Concurrent Transaction and Query Processing," *Proc. Second Int'l Workshop Research Issues on Data Eng.: Transaction and Query Processing*, pp. 12–19, 1992.
[18] C. Mohan, H. Pirahesh, and R. Lorie, "Efficient and Flexible Methods for Transient Versioning of Records to Avoid Locking by Read-Only Transactions," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 124–133, 1992.
[19] K.-L. Wu, P.S. Yu, and M.-S. Chen, "Dynamic Finite Versioning: An Effective Versioning Approach to Concurrent Transaction and Query Processing," *Proc. Int'l Conf. Data Eng.*, pp. 577–586, 1993.
[20] A. Merchant, K.-L. Wu, P.S. Yu, and M.-S. Chen, "Performance Analysis of Dynamic Finite Versioning for Concurrent Transaction and Query Processing," *Proc. 1992 ACM SIGMETRICS and PERFORMANCE '92*, pp. 103–114, 1992.
[21] T. Haerder and A. Reuter, "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, Dec. 1983.
[22] L. Kleinrock, *Queueing Systems, vol. I: Theory*. John Wiley & Sons, 1975.
[23] K. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science Applications*. Prentice-Hall, 1982.
[24] D.E. Knuth, *The Art of Computer Programming—Fundamental Algorithms*, vol. 1. Addison-Wesley, 1973.
[25] N.L. Johnson and S. Kotz, *Continuous Univariate Distributions-1*. Houghton Mifflin Co., 1970.
[26] S. Karlin and H.M. Taylor, *A First Course in Stochastic Processes*. Academic Press, 1975.
[27] E. Gumbel, *Statistics of Extremes*. Columbia Univ. Press, 1958.
[28] S. Ross, *Stochastic Processes*. John Wiley & Sons, 1982.
[29] P.M. Bober and D.M. Dias, "Storage Cost Tradeoffs for Multiversion Concurrency Control," Technical Report RC 18367, IBM T.J. Watson Research Center, Jan. 1992.



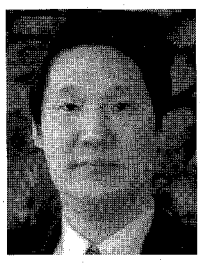
Arif Merchant received the BTech degree in computer science from the Indian Institute of Technology, Bombay, India, in 1984, and the PhD degree in computer science from Stanford University in 1991. He was with the IBM T.J. Watson Research Center, Yorktown Heights, New York from 1991 to 1992, and then joined the NEC Computer and Communications Research Laboratory in Princeton, New Jersey. Dr. Merchant is currently with the Storage Systems Program at Hewlett-Packard Laboratories in Palo Alto, California. His research interests include storage systems, multimedia server architectures, personal communications networks, and the applications of performance modeling to computer architecture.



Kun-Lung Wu (S'85-M'90) received the BS degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1982, and the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign in 1986 and 1990, respectively.

From 1985 to 1989, he was a research assistant at the Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois. In 1986, he also worked as a consultant at Texas Instruments Inc., Dallas, Texas. Since 1990, he has been with the IBM T.J. Watson Research Center, Yorktown Heights, New York, where he is currently a research staff member in the Architecture Analysis and Design Group. His current research interests include database transaction and query processing, performance analysis, internet application tools, memory and I/O management, multimedia applications, and network-centric information services.

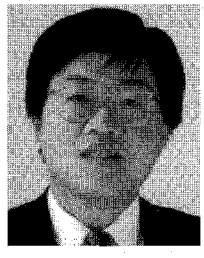
Dr. Wu has published many refereed journal/conference papers in his research areas. He is an inventor of patents in the areas of mobile computing, disk arrays, and concurrent transaction and query processing. He has also served as an organizing/program committee member for various IEEE conferences. He is a member of ACM and Phi Kappa Phi.



Philip S. Yu (S'76-M'78-SM'87-F'93) received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan, Republic of China, in 1972, the MS and PhD degrees in electrical engineering from Stanford University, in 1976 and 1978, respectively, and the MBA degree from New York University in 1982.

Since 1978, he has been with the IBM T.J. Watson Research Center, Yorktown Heights, New York. Currently he is manager of the Architecture Analysis and Design Group, which focuses on the design and analysis of clustering multiple processors for transaction and query processing. His current research interests include transaction and query processing, database systems, parallel and distributed processing, disk arrays, computer architecture, performance modeling, and workload analysis. He has published more than 200 papers in refereed journals and conferences, and more than 100 research reports and 80 technical disclosures. He holds or has applied for 33 US patents.

Dr. Yu is a member of the ACM and the IEEE. In addition to serving as program committee members on various conferences, he has served as the program chairman of the Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing. He has received several IBM and external honors including Best Paper Award, IBM Outstanding Innovation Award, Outstanding Technical Achievement Award, and 15 Invention Achievement Awards.



Ming-Syan Chen (S'87-M'88-SM'93) received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan in 1982, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1985 and 1988 respectively.

Dr. Chen is currently a faculty member in the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan. His research interests include database systems, multimedia technologies, and internet applications. He was a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, New York, from 1988 to 1996., primarily involved in projects related to parallel databases, multimedia systems, and data mining. He has published more than 70 refereed international journal/conference papers in his research areas/

Dr. Chen served as a guest co-editor of *IEEE Transactions on Knowledge and Data Engineering* on a special issue for data mining in 1996. He is the inventor of many international patents in the areas of interactive video playout, video server design, interconnection networks, and concurrency and coherency control protocols. He received the Outstanding Innovation Award from IBM in 1994 for his contribution to parallel transaction design for a major database product, and numerous award for his inventions and patent applications. Dr. Chen is a senior member of the IEEE and a member of the Association for Computing Machinery.

Authorized licensed use limited to: National Taiwan University. Downloaded on January 16, 2009 at 01:20 from IEEE Xplore. Restrictions apply.