Analysis of Optimistic Concurrency Control Revisited

Theodore Johnson

September 3, 1992

Abstract

Ryu and Thomasian have analyzed several varieties of optimistic concurrency control under two assumptions: that the characteristics of a transaction remain constant throughout its execution, and that the proportion of committing transactions is fixed (in order to avoid an unrealistic biasing of the transaction mixture). We show how both of these assumptions can be relaxed. First, we allow an aborted transaction to resample its execution time from the distribution that describes the transaction class execution times. Second, we allow transactions to change their class if they are aborted. These assumptions are more appropriate for analyzing special applications of optimistic concurrency control, such as nonblocking data structures and real-time optimistic transaction processing. We use our analyses to show that OCC performance has different characteristics in the new transaction models, and to analyze a simple nonblocking queue algorithm.

1 Introduction

Many types of database concurrency control have been proposed, including static locking, two-phase locking, certification, multiversion concurrency control, and optimistic concurrency control [2]. In this paper, we analyze optimistic concurrency control (OCC) [14]. Optimistic concurrency control is so named because it makes the optimistic assumption that data conflicts are rare. A transaction accesses data without regard to possible conflicts. If a data conflict does occur, the transaction is aborted and restarted.

Menasce and Nakanishi [17] present a Markov chain model of OCC in which aborted transactions leave, then reenter the transaction processing system as new transactions. Morris and Wong [18, 19] note that generating new transactions to replace aborted ones biases the transaction processing system towards executing short fast transactions. These authors provide an alternative solution method that avoids the bias by requiring that the transaction that replaces the aborted transaction be identical to the aborted transaction. Ryu and Thomasian [24] extend this model of OCC to permit a wide variety of execution time distributions and a variety of OCC execution models. Yu et al. [32, 31] develop approximate models of OCC and locking concurrency control to evaluate their performance in transaction processing systems.

```
read(g,Ta)
        read g into T's local workspace
        access_time(g)=Global_time


validate(T)
        for each g ∈ R(T)
                if access_time(g)<t(g)
                        abort(T)
        for every g ∈ W(T)
                t(g)=Global_time
        commit(T)
```

Figure 1: OCC validation

In this paper, we extend the simple but flexible model of Ryu and Thomasian [24] to handle additional transaction processing models. We modify the transaction model to examine the bias towards fast transactions that OCC introduces (taking the transaction execution model to that in [17]). We developed these OCC models as a tool for analyzing the performance of non-blocking data structure algorithms [20, 21, 29, 30], and they also have applications in analyzing real time optimistic transaction processing schemes [6, 9].

2 Model Description

Data access conflicts in OCC are detected by the use of *timestamps*. Each *data granule*, g, (the smallest unit of concurrency control) has an associated timestamp, $t(g)$, which contains the last time that the data granule was written to. Each transaction, T, keeps track of its read set $R(T)$ and write set $W(T)$. We assume that $R(T) \supset W(T)$. Every time a new data granule is accessed, the time of access is recorded. If at the commit point a data granule has a last write time greater than the access time, the transaction is aborted. Otherwise, the transaction is committed and the last write time of each granule in $W(T)$ is set to the current time. The procedure used is shown in figure 1.

Following Ryu and Thomasian, we distinguish between *static* and *dynamic* concurrency control. In static concurrency control, all data items that will be accessed are read when the transaction starts. In dynamic concurrency control, data items are read as they are needed. We also distinguish between *silent* and *broadcast* concurrency control. The pseudo-code in Figure 1 is silent optimistic concurrency control:

an operation doesn't advertise its commit, and transactions that will abort continue to execute. Alternatively, a transaction can *broadcast* its commit, so that conflicting transactions can restart immediately [22, 5].

We model the transaction processing system as a closed system in which V transactions each execute one of C transaction types. When a new transaction enters the system, it is a class c transaction with probability f_c, $\sum_C f_c = 1$. A class c transaction is assumed to have an execution time of $\beta(V)b_c(x)$, where $\beta(V)$ is the increase in execution time due to resource contention. Factoring out $\beta(V)$ is an example of a *resource contention decomposition approximation* [26, 24, 10], which lets us focus on the concurrency control mechanism, and which allows the analysis to be applied to different computer models. We will assume that $\beta(V) = 1$ in the analysis. Expansions in execution time can be factored back in after performing the concurrency control analysis.

As a transaction $\mathcal{T}$ executes, other transactions will commit their executions. If a committing transaction conflicts with $\mathcal{T}$, then $\mathcal{T}$ must be aborted. We denote by $\Phi(k, c)$ the probability that a committing class k transaction conflicts with an executing class c transaction. We model the stochastic process in which committing transactions conflict with an executing transaction as a Poisson process. Ryu and Thomasian [24] show that this assumption, which makes the analysis tractable, leads to accurate model predictions under a wide variety of conditions.

We differentiate between three models depending on the actions that occur when a transaction aborts. In [24], a transaction samples its execution time when it first enters the system. If the transaction is aborted, it is executed again with the same execution time as the first execution time. We call this transaction model the *fixed time/fixed class* model, or the FF model[1]. The FF model avoids a bias for fast transactions, permitting a fair comparison to lock-based concurrency control.

The variability of the execution time of a transaction could be due to resource contention, to decisions the transaction makes when as it accesses the database., or a combination of both. In these cases, the execution time of a transaction changes when a transaction is re-executed after an abort. We introduce the *variable time/fixed class*, or VF, model to represent this situation. In the VF model, an aborted transaction chooses a new execution time for its next execution.

There are some situations in which an aborted transaction simply exits the system and is not re-executed (for example, the transactions might be sensor data reports in a real time data monitoring

[1] Most of the results that we present for the FF model have been taken from [24].

system). In the *variable time/variable class*, or VV model, a new transaction type is picked to replace an aborted transaction (possibly a transaction of the same type).

2.1 Model Solution Methods

For a given transaction model, we can solve the system for any of the OCC models in the same way. The method for solving the system depends on the transaction model: the FF and the VF models use the same method, but the VV model is solved by using a different method.

2.1.1 Solving the FF and VF Models

The solution method for the FF and VF models involves taking the system utilization U (the portion of time spent doing useful work) and finding the per-class utilizations U_c. The system utilization U is then computed from the per-class utilizations. Ryu and Thomasian show that the equations can be solved quickly through iteration.

The mean useful residence time of a class c transaction is denoted by $R_a^c(V)$. A transaction might be required to restart several times due to data conflicts. The expected time that a transaction spends executing aborted attempts is denoted by $R_c^d(V)$, and the total residence time of a class c transaction is $R^c(V) = R_a^c(V) + R_d^c(V)$. The *utilization* of a class is the proportion of its expected residence time spent in an execution that commits: $U_c = R_A^c(V)/(R_A^c(V) + R_A^d(V))$. The expected residence time a transaction in the system is calculated by taking the expectation of the per-class expected residence times. The system efficiency is calculated by

$$U(V) = R_a(V) / \left(\sum_{c=1}^{C} f_c R_a^c(V)/U_c(V) \right) \tag{1}$$

In order to calculate the per-class efficiencies, we need to calculate the probability that a transaction aborts due to a data conflict. We define $\Phi(k, c)$ to be the probability that a class k transaction conflicts with a class c transaction. We know the proportions of committing transactions, so we can calculate the probability that a committing transaction conflicts with a class c transaction Φ_c by:

$$\Phi_c = \sum_{k=1}^{C} \Phi(k, c) f_k \tag{2}$$

We can calculate the rate at which a committing transactions conflict with a class c transaction, γ_c, by

setting γ_c to be the proportion of committing transactions that conflict with a class c transaction:

$$\gamma_c = \frac{(V-1)\Phi_c}{b} U(V)$$

where b is the expected execution time of all transactions.

Given the system utilization, we can calculate the per-class conflict rate. From the per-class conflict rate, we can calculate the per-class utilizations, and from the per-class utilizations, we can calculate the system utilization. The output system utilization is a decreasing function of the input system utilization. In the FF model, the utilization is bounded by 1, so the unique root in [0..1] can be found using a binary search iteration. In the VF model, it is possible for the utilization to be greater than 1, so the root finder must use one of the standard nonlinear equation solution methods [1].

2.1.2 Solving The VV Model

In the VV transaction model, when a transaction aborts, it leaves the system and a new transaction enters. As a result, the proportion of committing class c transactions is no longer f_c, and instead depends on the probability that a class c transaction commits, p_c, and the average execution time of a class c transaction. The solution method for the VV model is based on iteratively finding a root for the vector $\vec{p}$.

In order to calculate the conflict rate, we need to know the proportion of transactions S_k that are executing a class k transaction. When a process is executing a class k transaction, it executes for an expected b_k seconds. If one was to observe a very large number of transaction executions, say M, then a class k transaction would be executed about Mf_k times. Thus, the observation period would take $\sum_{i=1}^{C} Mf_ib_i$ seconds, during which a class k transaction would be executed for Mf_kb_k seconds. By the theory of alternating renewal processes [23], we have

$$S_k = f_kb_k / \sum_{i=1}^{C} f_ib_i \tag{3}$$

If process is executing a class k transaction, it will finish at rate $1/b_k$. When the transaction completes, it will commit at rate p_k, and if it commits, it will conflict with a class c transaction with probability $\Phi(k,c)$. Therefore,

$$\begin{aligned} \gamma_c = & \quad (V-1)\textstyle\sum_{i=1}^{C} S_kp_k\Phi(k,c)/b_k \\ = & \quad (V-1)\textstyle\sum_{k=1}^{C}(f_kb_k/\sum_{i=1}^{C} f_ib_i)p_k\Phi(k,c)/b_k \end{aligned}$$

$$= \quad \frac{V-1}{b} \sum_{k=1}^{C} \Phi(k,c) f_k p_k \tag{4}$$

Given the probability that transactions of each transaction class commits, $\vec{p_c}$, we can calculate conflict rate γ_c for each transaction class. Given the conflict rate for a transaction class γ_c, we can calculate the probability that the transaction will commit p_c.

Unlike the case with the FF and the VF models, for the VV model, we need to iterate on a vector. We make use of a property of the system of equations to find a rapidly converging iterative solution: if F is the transformation $F(\vec{p}_{old}) = \vec{p}_{new}$, then $F(p_1, \ldots, p_c + \epsilon, \ldots, p_C) \leq F(p_1, \ldots, p_c, \ldots, p_C)$, where $\epsilon > 0$ and the vector relation $\leq$ refers to component-wise comparison. In other words, the Jacobian of F is strictly nonpositive. The algorithm that we use to find a solution of the VV calculates the i^{th} value of p_c to be $p_c^i = (p_c^{i-1} + F(\vec{p}^{\,i})_c)/2$.

3 Analysis

In this section, we present the calculations needed for solve the systems discussed in the previous section. For each of the four types of optimistic concurrency control, we present the calculation for each of the three transaction models.

3.1 Analysis of Silent/Static OCC

In this section, we examine the simplest OCC scheme. In the silent/static scheme, transactions access their entire data sets when they start their executions, and detect conflicts when they attempt to commit.

3.1.1 Fixed Time/Fixed Class

In [24], if a transaction executes for t seconds, then aborts, it will execute for t seconds when it restarts. If an operation requires t seconds, the probability that it will be commit is $e^{-\gamma t}$, since we assume that conflicts form a Poisson process. Therefore, the number of times that a class c transaction with running time t must execute has the distribution

$$P_e^c(k|t) = \quad (1 - e^{-\gamma_c t})^{k-1} e^{-\gamma_c t}$$

and has mean $e^{\gamma_c t}$. A class c transaction with running time t therefore has a mean residence time of $te^{\gamma_c t}$, and class c transactions have a running time of

$$R^c(V) = \quad \int_0^\infty te^{\gamma_c t} b_c(t) dt$$

$$= \quad -\mathcal{B}_c^1(-\gamma_c)$$

where $\mathcal{B}_c^1$ is the first derivative of the Laplace transform of $b_c(t)$ [13]. Finally, the per-class utilization can be calculated for the iteration to be

$$U_c = \quad R_a^c(V)/R^c(V) \tag{5}$$

$$= \quad -b_c/\mathcal{B}_c^1(-\gamma_c) \tag{6}$$

We note that $b_c(t)$ must be $o(t^{-1}e^{-\gamma_c t})$ for the integral to converge.

3.1.2 Variable time / Fixed Class

In the variable time/fixed class model, every time a class c transaction executes its running time is sampled from $b_c(t)$. Therefore, the unconditional probability that the operation commits is:

$$p_c = \quad \int_{t=0}^{\infty} e^{-\gamma_c t} b_c(t) dt \tag{7}$$

$$= \quad \mathcal{B}_c(\gamma_c) \tag{8}$$

The number of times that the operation executes has a geometric distribution, so an operation will execute $1/p_c$ times. The first $1/p_c - 1$ times the operation executes, it will be unsuccessful. Knowing that the operation is unsuccessful tells us that it probably required somewhat longer than average to execute, since slow operations are more likely to be aborted. Similarly, successful operations are likely to be faster. In particular, an operation will be successful only if it reaches its commit point before a conflict occurs, and will be unsuccessful only if a conflict occurs before it reaches its commit point. The distributions of the execution times of the successful and unsuccessful operations is given by taking order statistics [3]:

$$b_c^s(t) = \quad K_s e^{-\gamma_c t} b_c(t) \tag{9}$$

$$b_c^f(t) = \quad K_f (1 - e^{-\gamma_c t}) b_c(t) \tag{10}$$

where K_s and K_f are normalizing constants computed by

$$K_s = \left(\int_0^{\infty} e^{-\gamma_c t} b_c(t) dt \right)^{-1}$$

$$K_f = \left(\int_0^{\infty} (1 - e^{-\gamma_c t}) b_c(t) dt \right)^{-1}$$

If b_c^s and b_c^f are the expected values of $b_c^s(t)$ and $b_c^f(t)$, respectively, then the expected time to complete a class c operation is

$$R_c(V) = b_c^s + (1/p_c - 1)b_c^f \tag{11}$$

$$= b_c^s + p_c^{-1}(1 - p_c)b_c^f \tag{12}$$

We observe that we only need to calculate b_c^s, because

$$b_c = p_c b_c^s + (1 - p_c)b_c^f \tag{13}$$

so that by combining (11) and (13) we get:

$$R_c(V) = b_c/p_c \tag{14}$$

Therefore, we find that

$$U_c = R_c^a(V)/R_c(V)$$

$$= b_c^s/b_c$$

$$= p_c \tag{15}$$

We note that in the variable time model, the only restriction on the distributions $b_c(t)$ is that they have finite means.

3.1.3 Variable-speed Model

For the silent/static VV model, we calculate the conflict rate from formula (4) and the probability that a class c transaction commits from formula (8).

3.2 Analysis of Static/Broadcast OCC

In static/broadcast OCC, transactions access their entire data sets when they start execution, and abort whenever a conflicting transaction commits.

3.2.1 Fixed/Fixed

The probability that a transaction restarts is calculated in the same way as in the silent/static model, given the same conflict rate. The wasted time per transaction now has a truncated exponential distri-

bution:

$$b_w^c(\tau|t) = \frac{\gamma_c e^{-\gamma_c \tau}}{1 - e^{-\gamma_c t}}$$

As a result,

$$U_c(V) = \frac{\gamma_c b_c}{1 - \mathcal{B}_c(-\gamma_c)} \tag{16}$$

3.2.2 Variable/Fixed

The probability that a transaction commits, p_c, and the expected execution time of transactions that commit b_c^s are calculated in the same way as in the silent/static model. The execution time of the aborted transactions is different, since a transaction will abort after t if some other transaction conflicts with it t seconds after it starts, and it has not yet committed:

$$b_c^f(t) = K_c^f \left[\gamma_c e^{-\gamma_c t}(1 - B_c(t))\right]$$

where

$$\begin{aligned} K_c^f &= \frac{1}{\int_0^\infty \gamma_c e^{-\gamma_c t}(1 - B_c(t))dt} \\ &= \frac{1}{1 - \gamma_c \int_0^\infty e^{-\gamma_c t} \int_0^t b_c(\tau)d\tau dt} \\ &= \frac{1}{1 - \mathcal{B}_c(\gamma_c)} \end{aligned}$$

Since a conflict aborts a transaction early, we can not make use of equation (13) to simplify equation (11). Instead, we must actually calculate the expected values b_c^s and b_c^f:

$$\begin{aligned} b_c^s &= (1/p_c) \int_0^\infty t e^{-\gamma_c t} b_c(t)dt \\ &= -\mathcal{B}_c'(\gamma_c)/p_c \end{aligned} \tag{17}$$

$$\begin{aligned} b_c^f &= K_c^f \int_{t=0}^\infty \gamma_c t e^{\gamma_c t}(1 - B_c(t))dt \\ &= \frac{1}{1 - \mathcal{B}_c(\gamma_c)} \left(1/\gamma_c - \gamma_c \int_0^\infty t e^{-\gamma_c t} \int_0^t b_c(\tau)d\tau dt\right) \\ &= \frac{1}{1 - \mathcal{B}_c(\gamma_c)} \left(1/\gamma_c - \gamma_c \left(\frac{d}{ds}\mathcal{B}_c(s)/s\Big|_{s=\gamma_c}\right)\right) \\ &= \frac{1}{1 - \mathcal{B}_c(\gamma_c)} \left(1/\gamma_c + \mathcal{B}_c'(\gamma_c) - \mathcal{B}_c(\gamma_c)/\gamma_c\right) \\ &= \frac{1 + \gamma_c \mathcal{B}_c'(\gamma_c) - \mathcal{B}_c(\gamma_c)}{\gamma_c(1 - \mathcal{B}_c(\gamma_c))} \end{aligned} \tag{18}$$

Putting these formulae into equation (11) for $R_c(V)$, we find that

$$R_c(V) = \frac{1-\mathcal{B}_c(\gamma_c)}{\gamma_c \mathcal{B}_c(\gamma_c)} \tag{19}$$

and,

$$U_c(V) = \frac{b_c \gamma_c \mathcal{B}_c(\gamma_c)}{1-\mathcal{B}_c(\gamma_c)} \tag{20}$$

We note that if $b_c(t)$ has an exponential distribution, then $U_c = 1$. This relation can be used to directly solve a system where all execution times are exponentially distributed, or to simplify the calculations when some execution time distributions are exponentially distributed and some are not.

3.2.3 Variable/Variable

In the silent/static case, a class k transaction executes for an expected b_k seconds. In the broadcast/static case, a transaction terminates early if it is aborted. The average amount of time that a transaction spends executing a class k transaction, $\bar{b}_k$, is the weighted average of the execution time depending on whether or not the transaction commits. By using equations (17) and (18), we find that:

$$\begin{aligned} \bar{b}_k &= p_k b_k^s + (1-p_k) b_k^f \\ &= (1-\mathcal{B}_k(\gamma_k))/\gamma_k \end{aligned} \tag{21}$$

Therefore, the proportion of time that a process spends executing a class k transaction is

$$S_k = f_k \bar{b}_k / \sum_{i=1}^{C} f_i \bar{b}_i \tag{22}$$

and the conflict rate of a class c transaction is

$$\gamma_c = \frac{V-1}{\bar{b}} \sum_{k=1}^{C} \Phi(c,k) f_k p_k \tag{23}$$

where $\bar{b} = \sum_{i=1}^{C} f_i \bar{b}_i$. Given a conflict rate γ_c, we calculate p_c by using equation (8).

3.3 Analysis of Silent/Dynamic

In dynamic optimistic concurrency control, a transaction accesses data items as they are needed. A class c transaction that requests n_c data items has n_c+1 phases. As the transaction accesses more data items, it acquires a higher conflict rate. We redefine the conflict function Φ to model the different phases of

the transactions. If a class k transaction commits, it conflicts with a class c transaction in stage i with probability $\Phi(k,c,i)$. The probability that a committing transaction conflicts with a class c transaction in stage i is:

$$\Phi_{c,i} = \sum_{k=1}^{C} f_k \Phi(k,c,i) \tag{24}$$

The conflict rate for a class c transaction in stage i is:

$$\gamma_{c,i} = \frac{(V-1)\Phi_{c,i}}{b} U(V) \tag{25}$$

The amount of time that a class c transaction spends in stage i has the distribution $b_{c,i}(t)$ with mean $b_{c,i}$, and the average time to execute the transaction is $b_c = \sum b_{c,i}$.

3.3.1 Fixed/Fixed

As a transaction moves through different stages, it encounters different conflict rates. The conflict rate for a class c transaction is a vector:

$$\vec{\gamma_c} = (\gamma_{c,1}, \gamma_{c,2}, \ldots, \gamma_{c,n_{c+1}})$$

Similarly, the execution time of a class c transaction is a vector $\vec{x} = (x_1, x_2, \ldots, x_{n_c+1})$, where x_i is a sample from the distribution with density $b_{c,i}(x)$. The probability that a class c transaction aborts is therefore

$$P_G^c = 1 - e^{\vec{\gamma_c} \cdot \vec{x}}$$

By taking expectations over the times for the processing stages, Ryu and Thomasian find that

$$R^c(V) = -\left(\prod_{i=0}^{n_c} \mathcal{B}_c(-\gamma_{c,i}) \right) \sum_{i=0}^{n_c} \frac{\mathcal{B}_c^1(-\gamma_{c,i})}{\mathcal{B}_c(-\gamma_{c,i})}$$

3.3.2 Variable/Fixed

We use the same transaction model as in the Fixed/Fixed case. A transaction will commit only it completes every stage without conflict. We define $p_{c,i}$ to be the probability that a class c transaction completes the i^{th} stage without a conflict. We can calculate $p_{c,i}$ by using formula (8), and substituting $\mathcal{B}_{c,i}$ for $\mathcal{B}$ and $\gamma_{c,i}$ for γ_c. Given the $p_{c,i}$, we can calculate p_c by

$$\begin{aligned} p_c &= \prod_{i=1}^{n_c+1} p_{c,i} \\ &= \prod_{i=1}^{n_c+1} \mathcal{B}_{c,i}(\gamma_{c,i}) \end{aligned} \tag{26}$$

As in the case of silent/static concurrency control, the unconditional expected time spent executing a class c transaction is b_c, so that

$$U_c = p_c \tag{27}$$

3.3.3 Variable/Variable

For the VV model, we use formula (4), appropriately modified to calculate the conflict rates, and formula (26) to calculate p_c.

3.4 Dynamic/Broadcast

3.4.1 Fixed/Fixed

The analysis of dynamic/broadcast concurrency control under the fixed/fixed model uses a combination of the previously discussed techniques. Ryu and Thomasian show that

$$R_d^c = R_{d1}^c + R_{d2}^c$$

$$R_{d1}^c = \mathcal{B}_c^1(0) \sum_{k=0}^{n_c} \left[k \left(\prod_{i>k}^{n_c} \mathcal{B}_c(-\gamma_{c,i}) \right) (1 - \mathcal{B}_c(-\gamma_{c,k})) \right]$$

$$R_{d2}^c(V) = \sum_{k=0}^{n_c} \left[\frac{1}{\gamma_{c,k}} \left(\prod_{i>k}^{n_c} \mathcal{B}_c(-\gamma_{c,i}) \right) (\mathcal{B}_c(-\gamma_{c,k}) - 1 - \gamma_{c,k} b_c) \right]$$

3.4.2 Variable/Fixed

We can use formula (26) to calculate p_c. For each processing phase, we can use formulae (17) and (18) to calculate $b_{c,i}^s$ and $b_{c,i}^f$. If a transaction commits, then it successfully completed each phase, so that

$$b_c^s = \sum_{i=0}^{n_c+1} b_{c,i}^s \tag{28}$$

If a transaction fails to commit, then it might have failed at any one of the $n_c + 1$ stages. We define $q_c = 1 - p_c$ to be the probability that a transaction aborts, and $q_{c,i}$ to be the probability that a transaction aborts at stage i, given that it aborts. A transaction that aborts at stage i must have successfully completed the previous $i-1$ stages, and a transaction aborts at exactly one of the stages, so

$$q_{c,i} = \frac{1 - p_{c,i}}{q_c} \prod_{j=1}^{i-1} p_{c,j}$$

If a transaction aborts at stage i, then its expected execution time is:

$$b^f_{c,i} + \sum_{j=1}^{i-1} b^s_{c,j}$$

Therefore, b^f_c is the unconditional expected execution time:

$$b^f_c = \sum_{i=1}^{n_c+1} \left((1 - p_{c,i}) \prod_{j=1}^{i-1} p_{c,j} \right) \left(\sum_{j=1}^{i-1} b^s_{c,j} + b^f_{c,i} \right) \tag{29}$$

We then use formulae (28) and (29) in formula (11) to find $R_c(V)$.

3.4.3 Variable/Variable

We use formula (26) to calculate p_c, and formulae (28) and (29) in formulae (21) and (23) to calculate the conflict rate.

4 Model Validation and Experiments

We wrote an OCC simulator to validate our analytical models. A parameterized number of transactions executed concurrently, and committing transactions conflicted with other transactions depending on a sample from Φ. We ran the simulation for 10,000 transaction executions, then reported statistics on throughput, execution time, and commit probabilities.

Ryu and Thomasian have already validated the F/F model, so we present a validation only of the V/F and V/V models (we also simulated the F/F model, and found close agreement between the simulation and analysis). In our first validation study, we modeled a system with a single transaction type. If there is only one transaction type, the V/F and the V/V models are the same, so we present results for the V/F model only (we also ran simulations and analytical calculations for the V/V model, and obtained nearly identical results). We calculated Φ by assuming that the transactions randomly accessed data items from a database that contained $N = 1024$ data items, and that transactions with overlapping data sets conflict. Ryu and Thomasian provide the following formula for the probability that two access sets of size n and m overlap in a database with N data items:

$$\Psi(n, m|N) = 1 - \binom{N-n}{m} / \binom{N}{m}$$

We report the probability that a transaction commits for a variety of access set sizes and degrees of concurrency in Table 1. The execution times in the static concurrency control experiments and the phase

V		Static/Silent			Static/Broadcast		
access set size		4	16	32	4	16	32
5	sim	.9391	.6418	.4655	.9378	.5270	.2807
	ana	.9444	.6366	.4586	.9414	.5272	.2797
15	sim	.8399	.4249	.2735	.8113	.2439	.0997
	ana	.8446	.4272	.2822	.8212	.2416	.0999
25	sim	.7716	.3474	.2152	.7201	.1569	.0614
	ana	.7755	.3481	.2241	.7281	.1567	.0608
		Dynamic/Silent			Dynamic/Broadcast		
5	sim	.9700	.7020	.4404	.9686	.6811	.3937
	ana	.9704	.7189	.4879	.9700	.6967	.4325
15	sim	.9025	.4587	.2627	.9039	.4187	.1971
	ana	.9071	.4733	.2627	.8479	.3071	.1319
25	sim	.8481	.3588	.1645	.8405	.2988	.1156
	ana	.8554	.3703	.1910	.8479	.3071	.1319

Table 1: Validation study of the V/F model. p_c is reported for a single transaction class and exponentially distributed execution times.

V	Varying/Fixed				Varying/Varying			
	analytical		simulation		analytical		simulation	
	class 1	class 2	class 1	class 2	class 1	class 2	class 1	class 2
5	.9692	.8495	.9684	.8941	.9692	.8946	.9672	.8897
15	.9069	.7081	.9069	.7088	.9066	.7073	.9009	.6919
25	.8589	.5864	.8552	.5797	.8574	.5828	.8511	.5628
35	.8203	.5011	.8167	.4906	.8168	.4935	.8014	.4660
45	.7884	.4379	.7886	.4282	.7822	.4263	.7675	.3986
55	.7576	.3812	.7612	.3892	.7521	.3736	.7294	.3495

Table 2: Validation study for Dynamic/Broadcast OCC and two transaction classes. p_c is reported. Execution phase times are exponentially distributed.

execution times in the dynamic concurrency control experiments were exponentially distributed. The experiments show close agreement between analytical and simulation results, though the calculations are least accurate for the dynamic concurrency control when the level of conflict is high.

We also performed a validation study for a system with two transaction classes. The first transaction class accesses four data items, and the second accesses eight data items. To save space, we reports results for dynamic/broadcast OCC only. Table 2 reports simulation and analytical results for the V/F and the V/V transaction models for a variety of degrees of concurrency. In these experiments, $f_1 = .6$ and $f_2 = .4$. We found close agreement between the simulation and the analytical predictions.

	SS		SB		DS		DB	
V	FF	VF	FF	VF	FF	VF	FF	VF
5	4.012	4.102	4.420	4.529	4.425	4.482	4.593	4.653
10	6.621	6.899	7.780	8.168	7.829	8.036	8.379	8.610
15	8.582	9.075	10.479	11.223	10.604	11.016	11.596	12.074
20	10.163	10.875	12.728	13.864	12.957	13.603	14.391	15.150
25	11.482	12.419	14.651	16.196	15.002	15.894	16.861	17.937
30	12.637	13.778	16.329	18.285	16.811	17.962	19.072	20.484

Table 3: Comparison of OCC performance, Erlang distribution. Throughput is reported

4.1 Comparison of OCC Schemes

We ran a set of experiments to determine the effect of the different OCC schemes on system performance. In table 3, we compare the performance of SS, SB, DS, and DB OCC for the FF and the VF transaction models. The transaction processing system contains a single transaction class, which accesses 8 data items in a database of 1024 data items. The total execution time for all experiments has an Erlang distribution with mean 1.0.

We find the surprising result that SB concurrency control gives better performance than DS concurrency control in the VF transaction model. The FF transaction model give the opposite result. For the VF model, SS $<$ DS $<$ SB $<$ DB, while in the FF model, SS $<$ SB $<$ DS $<$ DB. We also find that a transaction processing system with VF transactions has a higher throughput than a transaction processing system with FF transactions (though the transaction model is typically fixed by the application).

We also ran experiments for SS and SB concurrency control in which the transaction execution times are exponentially distributed. These results are listed in table 4. We found that while the performance of the FF transactions decreased with the increase in execution time variance, the performance of the VF transactions increased. The performance of the SB OCC on the VF transactions became significantly greater, surpassing the performance of the DB OCC.

5 An Application: Nonblocking Data Structures

Many parallel data structures use locks to ensure simultaneous but non-interfering access. Other parallel data structures avoid the use of locks [16, 20, 21, 15, 7, 8, 25, 27, 4, 28, 29, 30]. *Nonblocking* data structures avoid the use of locks, and also guarantee that no operation blocks the execution of any other operation for more than a finite number of steps [7]. Nonblocking data structures are tolerant of

	SS		SB	
V	FF	VF	FF	VF
5	3.456	4.157	4.019	5.000
10	5.146	7.174	6.455	10.000
15	6.237	9.672	8.049	15.000
20	7.026	11.854	9.262	20.000
25	7.636	13.817	10.140	25.000
30	8.129	15.616	10.830	30.000

Table 4: Comparison of OCC performance, exponential distribution. Throughput is reported

process slow-downs and failures, and can also provide better performance than blocking data structures by permitting fast operations to occur at the expense of slow operations.

A nonblocking data structure avoids the use of locks by using optimistic concurrency control instead (relying on the use of the compare-and-swap instruction). As a result, the performance of nonblocking data structures can't be analyzed by using the analytical tools developed for locking data structures [10, 11, 12]. This work was motivated by the desire to develop tools for analyzing nonblocking data structures.

We developed a simple nonblocking queue that permits an enqueue to occur in parallel with a dequeue operation [20]. We wrote a detailed simulation to compare the performance of the nonblocking queue to a locking queue. We modeled the processors in the parallel computer as being either *constant speed* or *varying speed*. In the constant speed model, the processors are divided into fast and slow processors. When an operation enters to compete for the queue, it is assigned a fast or a slow processor with a fixed probability. In the varying speed model, all processors are identical, but the processors cycle between being slow or fast. When an operation enters the system, it is assigned a processor, which can be in either the fast or in the slow state.

Our simulation results showed the surprising result that nonblocking queue has worse performance than the blocking queue in the constant speed model, but better performance in the varying speed model [21]. In order to examine this phenomenon more closely, we analyze the queue using the V/F and the V/V models.

In the nonblocking queue, the enqueue operations are serialized with respect to each other, and so are the dequeue operations, but an enqueue operation conflicts with a dequeue operation only when the queue is empty. We model the enqueue stream only and assume that the queue never becomes empty.

Therefore, every operation conflicts with every other, so $\Phi = 1$. We use two transaction classes to model the fast and slow processors. The first transaction class models the fast processors. Its execution time is chosen uniformly randomly in $[.8, 1.2]$, and $f_1 = .9$. The execution time of the second transaction class, which represents the slow processors, is chosen uniformly randomly in $[8, 12]$, and $f_2 = .1$. The VF transaction model represents the case in which the operation chooses a fast or a slow processor when it begins execution, and the VV model represents the case in which the processor that is executing the operation is usually fast but occasionally becomes slow.

We plot the throughput of the nonblocking queue for the constant speed and the varying speed models against increasing V in figure 2. For comparison, we also plot the throughput of the locking algorithm algorithm, which is a constant $1/b = 1/1.9$. The nonblocking queue in the constant speed model has a lower throughput than the locking queue, in spite of the preference shown towards fast executions in the VF model. The nonblocking queue has a poor throughput in the constant speed model because of the extremely long times required for the completion of the operations executed on the slow processors. These running times are shown in figure 3. The throughput of the variable speed model increases with increasing V, and is considerably greater than that of the locking queue. These model predictions are in agreement with our simulation results [21].

6 Conclusion

In this paper, we have extended the OCC performance model due to Ryu and Thomasian (which we call the FF model) to handle transactions that resample their execution time if they abort (VF transaction model), and also to handle transactions that change their transaction class (VV transaction model). We present performance models for static/silent, static/broadcast, dynamic/silent, and dynamic/broadcast optimistic concurrency control. We validate each of the models by comparison to simulation results.

We investigate the performance of OCC under the VF and VV models and find that the throughput of the transaction processing system increases as the variance in the execution times increase In contrast, the throughput of a transaction processing system running FF transactions decreases with increasing variance in execution times. We compare the performance of the different OCC schemes for the VF model, and find that SS $<$ DS $<$ SB $<$ DB in terms of throughput. The FF model orders the OCC schemes differently, giving SS $<$ SB $<$ DS $<$ DB in terms of throughput. Therefore, if the VF model applies to he transactions, it is better to implement broadcast optimistic concurrency control than it is

to implement dynamic concurrency control.

We apply the new OCC performance models to the analysis of nonblocking data structures. We find that nonblocking data structures perform worse than locking data structures in a NonUniform Memory Architecture (NUMA) architecture, but better than locking data structures in a Uniform Memory Architecture (UMA) architecture.

References

[1] K.E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley and Sons, 1978.

[2] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[3] H.A. David. *Order Statistics*. John Wiley, 1981.

[4] A. Gottlieb, B. D. Lubachevsky, and L. Rudolph. Coordinating large numbers of processors. In *Proceedings of the International Conference on Parallel Processing*. IEEE, 1981.

[5] T. Harder. Observations on optimistic concurrency control schemes. *Inform. Systems*, 9(2):111–120, 1984.

[6] J. Haritsa. Transaction scheduling in firm real-time database systems. Technical Report TR1036, University of Wisconson-Madison, Dept. of CS, 1991.

[7] M. Herlihy. A methodology for implementing highly concurrent data structures. In *Proceeding of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 197–206. ACM, 1989.

[8] M. Herlihy and J. Wing. Axioms for concurrent objects. In *Fourteenth ACM Symposium on Principles of Programming Languages*, pages 13–26, 1987.

[9] J. Huang, J. Stankovic, K. Ramamritham, and D. Towsley. Experimental evaluation of real-time optimistic concurrency control schemes. In *Proc. 17th VLDB*, 1991.

[10] T. Johnson. *The Performance of Concurrent Data Structure Algorithms*. PhD thesis, NYU Dept. of Computer Science, 1990.

[11] T. Johnson and D. Shasha. A framework for the performance analysis of concurrent B-tree algorithms. In *ACM SIGACT/SIGMOD/SIGART Symposium on Principles of Database Systems*, pages 273–287, 1990.

[12] T. Johnson and D. Shasha. The performance of concurrent data structure algorithms. *Transactions on Database Systems*, 1992. to appear.

[13] L. Kleinrock. *Queueing Systems*, volume 1. John Wiley, New York, 1975.

[14] H.T. Kung and J.T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981.

[15] L. Lamport. Specifying concurrent program modules. *ACM Trans. on Programming Languages and Systems*, 5(2):190–222, 1983.

[16] V. Lanin and D. Shasha. Concurrent set manipulation without locking. In *Proceedings of the Seventh ACM Symposium on Principles of Database Systems*, pages 211–220, 1988.

[17] D. Menasce and T. Nakanishi. Optimistic vs. pessimistic concurrency control mechanisms in database management systems. *Information Systems*, 7(1):13–27, 1982.

[18] R. Morris and W. Wong. Performance of concurrency control algorithms with non-exclusive access. In *Performance '84*, pages 87–101, 1984.

[19] R. Morris and W. Wong. Performance analysis of locking and optimistic concurrency control algorithms. *Performance Evaluation*, 5:105–118, 1985.

[20] S. Prakash, Y.H. Lee, and T. Johnson. A non-blocking algorithm for shared queues using compare-and-swap. In *Proc. Int'l Conf. on Parallel Processing*, pages II68–II75, 1991.

[21] S. Prakash, Y.H. Lee, and T. Johnson. Non-blocking algorithms for concurrent data structures. Technical report, University of Florida Dept. of CIS, 1991. Available at cis.ufl.edu:cis/tech-reports/tr91/tr91.002.ps.Z.

[22] J.T. Robinson. Experiments with transaction processing on a multiprocessor. Technical Report RC9725, IBM, Yorktown Heights, 1982.

[23] S.M. Ross. *Stochastic Processes.* John Wiley, 1983.

[24] I.K. Ryu and A. Thomasian. Performance analysis of centralized database with optimistic concurrency control. *Performance Evaluation*, 7:195–211, 1987.

[25] J. Stone. A simple and correct shared-queue algorithm using compare-and-swap. Technical Report RC 15675, IBM TJ Watson Research Center, 1990.

[26] Y.C. Tay, R. Suri, and N. Goodman. Locking performance in centralized databases. *ACM Transactions on Database Systems*, 10(4):415–462, 1985.

[27] R. Treiber. Systems programming: Coping with parallelism. Technical Report RJ 5118, IBM Almaden Research Center, 1986.

[28] J. Turek. *Resilient Computation in the Presence of Slowdowns.* PhD thesis, NYU Dept. of Computer Science, 1991.

[29] J.D. Valois. Analysis of a lock-free queue. Submitted for publication, 1992.

[30] J.D. Valois. Concurrent dictionaries without locks. Submitted for publication, 1992.

[31] P.S. Yu, D.M. Dias, and S.S. Lavenberg. On modeling database concurrency control. Technical Report RC 15368, IBM Research Division, 1990.

[32] P.S. Yu, H.U Heiss, and D.M Dias. Modeling and analysis of a time-stamp history based certification protocol for concurrency control. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):525–537, 1991.

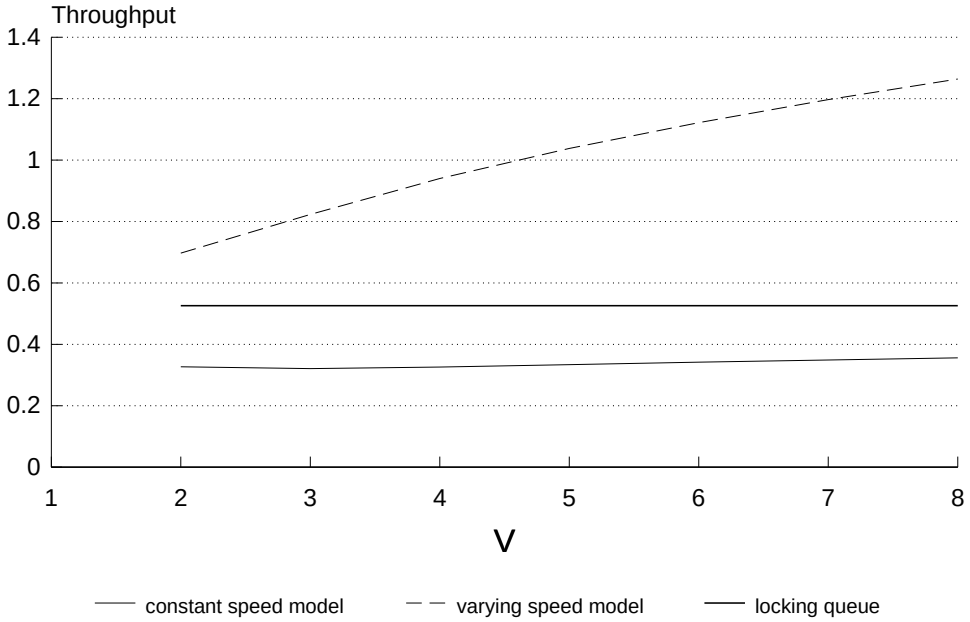

Figure 2: Throughput of the locking queue and the nonblocking queue in the constant speed and the varying speed model

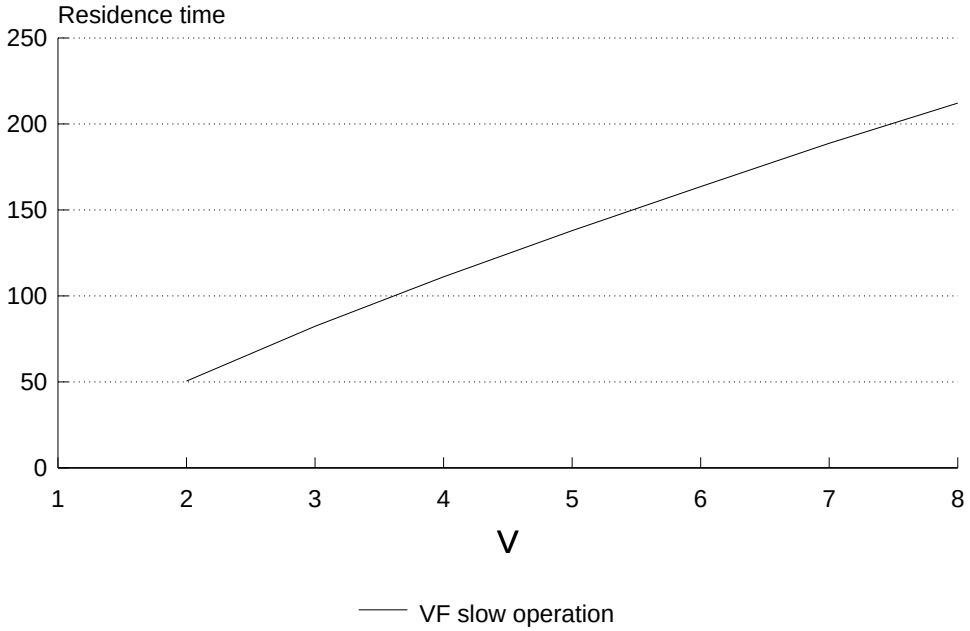

Figure 3: Response time of the slow operations in the constant speed model