A Comparative Study of Divergence Control Algorithms

Akira Kawaguchi and *Kui Mok*
Department of Computer Science
Columbia University
New York, NY 10027

Calton Pu
Dept. of Computer Science & Engineering
Oregon Graduate Institute
P.O. Box 91000, Portland, OR 97291-1000

Kun-Lung Wu and *Philip S. Yu*
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

CUCS-013-94

Abstract

This paper evaluates and compares the performance of two-phase locking divergence control (2PLDC) and optimistic divergence control (ODC) algorithms using a comprehensive centralized database simulation model. We examine a system with multiclass workloads in which on-line update transactions and long-duration queries progress based on epsilon serializability (ESR). Our results demonstrate that significant performance enhancements can be achieved with a non-zero tolerable inconsistency (ϵ-spec). With sufficient ϵ-spec and limited system resources, both algorithms achieve comparable performance. However, with low resource contention, ODC performs significantly better than 2PLDC. Moreover, given a small ϵ-spec, ODC returns more accurate results on the committed queries then 2PLDC.

Index Terms: divergence control algorithms, epsilon serializability, performance analysis, transaction and query processing.

1 Introduction

Serializability (SR) is maintained by concurrency control (CC) algorithms [3] in online transaction processing. Many applications have found SR too restrictive, and epsilon serializability (ESR) was proposed to alleviate SR constraints by allowing some bounded inconsistency. In particular, a limited amount of inconsistency (ϵ-spec) can be seen by read-only transactions. Divergence control (DC) algorithms have been designed [18, 14] to bound the amount of inconsistency for ESR the same way concurrency control maintains SR. In fact, DC algorithms extend classic CC algorithms such as two-phase locking, optimistic validation, and timestamps [11, 15].

Some important questions about ESR have been answered by previous papers. For example, a formal characterization of ESR [15] explains its meaning and relationship to SR. Also, several papers have described the divergence control algorithms used to guarantee ESR [11, 14, 18]. The question on practical feasibility was answered by a simple implementation of two-phase locking based divergence control built on top of Transarc Encina [12], a commercial transaction monitor. These and other ESR papers have worked out the details of divergence control and shown the feasibility of quick production quality implementation. However, except for one paper studying the performance of hierarchical divergence control using timestamps [9], the quantitative behavior of ESR systems remained unclear.

The main contributions of this paper, compared to the previous ESR work, are threefold. First, we investigate the question of how much inconsistency must be tolerated before we see significant performance gains in terms of transaction throughput and response time. We provide answers to a wide range of representative system and workload parameters. Second, we evaluate and compare the performance of two-phase locking divergence control and optimistic divergence control, finding the strengths and weaknesses of both. Third, we explore the design space of divergence control algorithms. This investigation found interesting system and workload parameters useful in guiding the actual implementation of divergence control and its experimental evaluation.

Concretely, we explored the issue of ESR applicability to large-scale centralized TP environment. A comprehensive simulation model is built to determine transaction throughput and response time. The first quantitative performance evaluation [9] was based on a prototype using time stamp ordering on both transaction and object levels in a hierarchical manner. Despite the limitation of main memory database and small range of multiprogramming level (MPL), their paper clearly demonstrated the shift of thrashing points to a higher MPL as ϵ-spec increases. Our simulation program showed significant performance improvement by both two-phase locking (2PLDC) and optimistic (ODC) algorithms in a wider range of MPLs as inconsistency tolerance increases. Our study further leads to a precise study on the distribution of inconsistency. Queries are found to be more accurate in ODC than in 2PLDC, with a small inconsistency tolerance.

The rest of the paper is organized as follows. Section 2 briefly reviews ESR and DC algorithms. In Section 3, we describe the performance model and outline the experiments. We present and discuss performance results in Section 4. Finally, we summarize the main conclusions of this study in Section 5.

2 Divergence Control Algorithms

A formal characterization of ESR can be found in [15]. We only informally introduce the basic ESR concepts here. A classic transaction is extended to an epsilon-transaction (ET) by the addition of inconsistency limits, called an ϵ-spec. An ET is allowed to commit if the inconsistency accumulated by the ET during its execution is lower than its ϵ-spec specified by the application designer. For example, when ϵ-spec=0 ETs reduce to atomic transactions. Another example is a bank summary query, which tells how many millions of dollars the bank has; since the query's answer unit may contain up to half a million round-off error, the query can tolerate a certain amount, say $100,000, of bounded inconsistency.

There are two kinds of ETs: QET for query ET and UET for update ET. A QET contains a sequence of one or more read-only (Q) operations. An UET contains a mixture of read (R) and write (W) operations in which R precedes W for any update operation as [1]. A QET (import ϵ-spec > 0) need not be serializable with other UETs whereas UETs (export ϵ-spec > 0 but import ϵ-spec=0) must be serializable among themselves. Each W alters the database state. Database state spaces can be metric spaces. In particular, bank dollars (integers) form a cartesian space. Consequently, each W operation has an update amount, given by the distance between the old state and the new state. ESR allows non-serializable behavior. Note that ESR never introduce inconsistency to the database space because all database modifications (UETs) are serializable.

In example (1) below, each Q operates on a data item denoted by ρ_i. In example (2), W updates ρ_i with a distance between the old value and new value denoted by Δ_i^W.

(1) QET($Q(\rho_1)$, $Q(\rho_2)$)

$$(2) \qquad \text{UET}(R(\rho_1),\ W(\rho_1 + \Delta_1^W),\ R(\rho_2),\ W(\rho_2 + \Delta_2^W),\ R(\rho_3))$$

Consider the following interleaving of the above example QET and UET:

$$Q(\rho_1), R(\rho_1), W(\rho_1 + \Delta_1^W), R(\rho_2), W(\rho_2 + \Delta_2^W), Q(\rho_2), R(\rho_3).$$

Suppose also that inconsistency is evaluated simply by adding update amounts. Then, the QET imported $|\Delta_1^W| + |\Delta_2^W|$ from the UET, and the UET is said to have exported the same amount.

DC algorithms take advantage of the fact that CC algorithms must detect any potentially non-serializable conflicts among transactions. When such conflicts are detected, a DC algorithm ensures that inconsistency accumulation for each ET does not violate its ϵ-spec. We call the CC and DC algorithms as *strict serialization* and *ESR extension* respectively. In the next two sections, we describe our implementation of strict two-phase locking divergence control [3] and broadcast-type optimistic divergence control [18, 20, 21] in the way they integrate strict serialization and ESR extension.

2.1 Two-phase Locking Divergence Control (2PLDC)

2PLDC is an extension of the classic 2PL concurrency control. We show a lock compatibility matrix in Figure 1(a)[1]. Suppose that DC initiates a process to schedule each ET. We call this process a *monitor*. For strict serialization, when a monitor finds a lock request created by a different monitor (i.e., the operation having been issued by another ET) such as a conflict of R and W, 2PLDC forces the current monitor to wait until the conflict is cleared, unless deadlock occurs.

For ESR extension, a conflict of Q and W between two ETs can be resolved only if inconsistency accumulation by the new update operation's amount does not exceed their ϵ-specs. For clarity, we denote a QET's ϵ-spec as ϵ-spec^Q and UET's as ϵ-spec^U. We also denote monitor^Q and monitor^U dealing with Q and W respectively. If the monitor^Q finds W or if the monitor^U finds Q in the lock table[2], the current monitor checks the following two conditions:

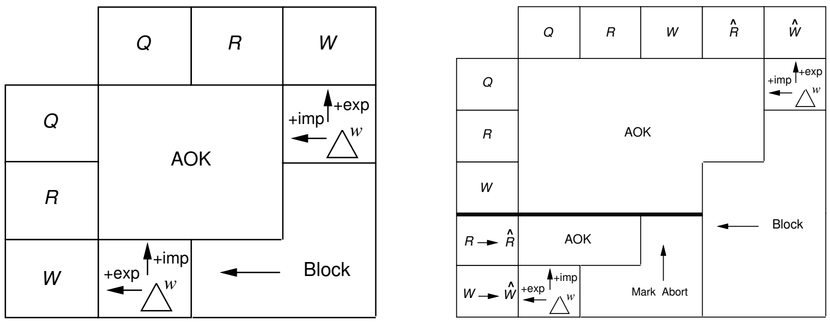
$$\text{imp-accum.monitor}^Q + \sum |\Delta^W| \le \epsilon\text{-spec}^Q;$$
$$\text{exp-accum.monitor}^U + \sum |\Delta^W| \le \epsilon\text{-spec}^U.$$

[1]AOK means always OK.

[2]The former is called LOK-1 (Limited OK), and the latter is called LOK-2.

If the conditions remain true, the operation is allowed to proceed, as if there were no conflicts. Note that the accumulation of values in *imp-accum* and *exp-accum* happens only if both conditions are met. If the conditions are violated, the action taken is the same as the corresponding strict serialization: blocking for 2PL and abort for ODC.

Commit occurs after the monitor executes the ET's last operation. All locks of the committing ET are released. Abort occurs when the monitor detects deadlock in a resource wait-for graph. Abort processing is essentially the same as the commit except that the ET is restarted.



(a) 2PLDC (b) ODC

Figure 1: Lock Compatibility Matrices

2.2 Optimistic Divergence Control (ODC)

Our optimistic validation divergence control method uses the weak lock/strong lock formulation, which is equivalent to a graph traversal algorithm to determine whether there is any cycles in the transaction dependence graph. In the weak lock scheme, data access happens after a weak lock is acquired. Conflicts between weak locks are allowed. At commit time, each ET attempts to convert its weak locks to strong locks. A conflict between strong and weak locks causes aborts.

We show the lock compatibility matrix in Figure 1(b). Every ET starts from the non-blocking phase and it *escalates* all locks associated with the past operations during the validation phase [20]. For strict serialization, when scheduling a new operation, a monitor investigates whether there exists escalated locks from other ETs. An escalation-flag indicates the midst of commit. If the monitor finds such one, for instance, when monitor^U finds $\hat{W}$, it waits for the commit completion. The monitor proceeds if there exists no conflict.

For ESR extension, the conflict between a Q and a committing $\hat{W}$ is allowed only if new inconsistency adjustments do not exceed their ϵ-specs. However, ESR extension also occurs in the opposite combination, i.e., at the commit moment when W is escalated to $\hat{W}$ and it encounters another Q as explained below. Now, if the monitor^Q finds $\hat{W}$ in the lock table[3], it checks the following two conditions:

$$\text{imp-accum.monitor}^Q + \sum|\Delta^{\hat{W}}| \leq \epsilon\text{-spec}^Q;$$
$$\text{exp-accum.monitor}^U + \sum|\Delta^{\hat{W}}| \leq \epsilon\text{-spec}^U.$$

Note that the replacement of values in *imp-accum* and *exp-accum* will be done only if both conditions are met for every conflict. Treatment right after this extension is the same as the strict serialization.

Commit occurs when the monitor finds no remaining actions. When monitor^U enters into the validation stage, it at first escalates the status of all pertinent locks. This escalation marks every other monitor^U abort if they are scheduled with non-escalated W on the same entity. The monitor^U having non-escalated R is also marked for abort when the committing ET escalates to $\hat{W}$ on the same entity.

If the committing monitor^U finds a conflict against Q, it needs to check both ϵ-specs as described in the ESR extension. The monitor^Q is marked for abort if the condition is unsatisfactory. UET also fails to commit if it exceeds its own inconsistency boundary. The same check occurs when the monitor^Q for QET enters into the committing stage, although no escalation for action Q takes place. Note again that values in accumulators are not updated until ETs commit. Finally, all locks of committing ETs are released, and aborted ETs are restarted after clearing all locks.

3 Centralized Database Performance Model

Our goal is to quantify the benefits gained by DC evolved from a traditional centralized TP (including a shared database with high-speed LAN and clustered machines) to the one with ESR. We thus built the TP performance model based on the framework introduced by [1, 4] which consists of database, user, and transaction models. In the following two sections, we outline the structure and implementation of our performance model, and provide a list of parameters used for experiments presented in this paper.

[3]Strictly speaking, this situation occurs when the committing process does context switch to another transaction, but this is not the case in our simulator.

3.1 Structure of Performance Model

Figure 2 shows three major components in our closed queuing model. First, *Transaction Controller* generates ETs in a probabilistic way that each terminal from a predefined number of pools in the system initiates a new ET when the previous one it issued returns. Each ET consists of an ordered list of operations that target on the smallest data entities. At the moment of generation, each terminal determines an ET (UET or QET), as well as its size. For UETs, the W following a R is determined by the conditional probability Pr(W|R).

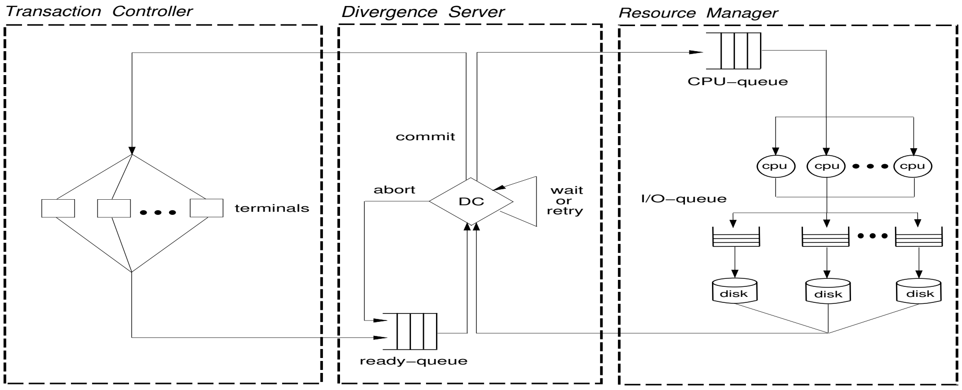


Figure 2: Centralized Database Performance Model

Second, the *Divergence Server* picks up ready transactions from the ready-queue. It first checks for strict serialization and further verifies ESR extension. It then issues a request to the *Resource Manager* where the CPU computation or disk I/O is evaluated subsequently. But for unsuccessful scheduling, it defers rescheduling until an appropriate moment, or aborts and puts the corresponding transaction back to the ready-queue. An aborted transaction is granted a higher priority from which a successive rescheduling is promptly done.

Third, the *Resource Manager* conducts both computation and disk I/O activities in a tightly connected multiprocessor system. As in [1], we define CPUs as multiple servers through a common queue, and each disk server has its own queue. In addition, we employ a data-buffering mechanism driven by LRU replacement policy. Both CPU and I/O requests are served based on a FCFS discipline, but CPU requests from DC have higher priority over other service requests.

The processing of an ET by the Resource Manager is as follows. Divergence Server forwards a request to Resource Manager and it waits until the request completes. Within Resource Manger, each service is simulated with deferred write policy; i.e., updates are written first into the buffer pool and then onto the disks at commit time. Reading an object requires disk I/O followed by

CPU. Writing an object requires CPU on the read item but it involves disk I/O for writing-back at commit. When the request is done, Divergence Server continues. Note that each Divergence Server also uses CPU for its scheduling. Thus, CPU usage is preemptive by priority. Divergence Server has a higher priority to use CPU, whereas each operation in ET has a lower priority. Thus, it can steal CPU being spent on the operation at Resource Manager whenever its scheduling is possible.

3.2 Experiments

Table 1 summarizes our workload and resource parameter settings. Our simulation program is implemented using CSIM [16, 17]. The simulation is flexible with a large number of parameters, which enables us to characterize the workload more precisely. For validation, we ran a set of experiments with parameters identical to those used in [1]. Furthermore, most of the values are kept unchanged from [1] to maintain the validation and basis for comparison.

3.2.1 Workload Parameters

Our workload consists of two classes of transactions: short updates (UETs) and lengthy queries (QETs). Unlike TPC benchmarks, where data contention is minimized, we introduce some interference between UETs and QETs. We assume that such combination is realistic in many transaction systems with decision support applications [7].

The size of an UET is decided by the mean of a uniform distribution between $tran\_Umin\_size$ and $tran\_Umax\_size$. Each UET chooses a target entity randomly from a database of *tran_db_size* entities. The percentage of a write operation on a read item is determined by *tran_write_prob*. The size of a QET is modeled by a fixed length of $tran\_Qmax\_size$. We draw a starting point in the database and generate QET read requests in an ascending order. The rate of QETs issued by each terminal is determined by *tran_query_prob*. A pause of the submission of a new transaction from a terminal is set by the exponential distribution with *tran_mean_thinktime*.

Since non-uniform data accesses always happen in real databases [5, 6, 7], we characterize data contention caused by skewed access patterns using $tran\_access\_frac$, where different frequencies of access relate to separate segments in the database. In our experiments, we choose an 80%-20% model, i.e., 80% of all accesses go to 20% of the database entities. We also distribute skewed hot-spots evenly in the database so that sequentially generated QETs will access these hotspots.

Finally, each W operation is associated with an update amount from a normal distribution between $tran\_min\_offset$ and $tran\_max\_offset$ with a standard deviation σ set to one sixth of this range. Note that the average amount of inconsistency between a Q and a W is $\sigma\sqrt{2/\pi}$.

Workload Parameter	**Value**	**System/Resource Parameter**	**Value**
tran_db_size	10,000 entities	*sys_num_terminal*	200
tran_mean_thinktime	1 sec	*sys_num_cpu*	10, ∞
tran_access_frac	80%-20%	*sys_num_disk*	20, ∞
tran_Umax_size	12 entities	*sys_mean_cpu*	15 ms
tran_Umin_size	4 entities	*sys_mean_disk*	35 ms
tran_write_prob	0.25	*buffer_size*	2,000
tran_Qmax_size	500	*buffer_UET_frac*	0.95
tran_query_prob	0.25	*buffer_algorithm*	LRU
tran_min_offset	-500	*sys_dc_cpu*	0 ms
tran_max_offset	500		

Table 1: Parameters

3.2.2 System and Resource Parameters

There are two sets of experiments, namely, resource-limited and resource-unlimited experiments. The resource-limited case studies the relationships between data contention and other system bottlenecks. The resource-unlimited case assumes infinite system resources and focuses on the impact of data contention only.

Physical configuration of the system is determined in a straightforward way. They include *sys_num_terminal* for external terminals and *sys_num_cpu* and *sys_num_disk* for the amount of CPU and disk servers respectively. As in [1], we use one CPU and two disks as a basic resource unit. We also inherit a constant service time spent on a single request to CPU and disk as *sys_mean_cpu* and *sys_mean_disk* as well as negligible *sys_dc_cpu* overhead in scheduling.

Our characterization of buffering effect is determined by *buffer_size* and *buffer_algorithm* in which *buffer_UET_frac* specifies area assigned for UETs. We limit 5% of the entire buffer for QETs because their sequential accesses take less advantage from buffering.

3.2.3 Performance Metrics

The three major performance metrics used in this study are *throughput*, *response time*, and *disk utilization* of the underlying system. All these statistics are drawn within a MPL range from 5 to 175. We additionally collect *abort/commit ratio* (average number of aborts per commit) and *accumulated inconsistency* for a QET.

The transaction throughput is defined as the total number of transactions successfully completed per second. The response time is the elapsed time between a terminal issuing a new transaction and the transaction actually being committed. Disk utilizations are useful to visualize the effect of resource contention on the performance of both 2PLDC and ODC. Abort/commit ratio is the average number of aborts (and retries) until the transaction commits.

Average update amount ($|\Delta^W|$) in our setting is 133. We set ϵ-spec to 1,000, 3,000 and 5,000 as well as an extreme value to approximate ∞ for both UETs and QETs. They are rather ad hoc but they range from several to a few dozens of update amounts, corresponding roughly from a small to a large tolerance in this context.

4 Results and Discussions

All statistics in the next three sections are derived using a batch means method [1]. A maximum of 20 batches were run on each simulation where each batch lasts 1,000 simulation seconds. The transient period of the initial 100 seconds is not used in the computation of the final statistics. Furthermore, we maintain the 90% confidence interval to be within 5% of each data point reported in this paper.

4.1 Experiments with Resource-Limited Environments

In this section, we investigate the combined performance impacts of both data and resource contentions. For resource contention, we set a limited number of CPUs and disks. Due to the space limitation, we only present the results of using 10 resource units (10 CPUs and 20 disks). Figure 3(a)–(d) show the throughputs and response times of UETs and QETs for both 2PLDC and ODC under various ϵ-specs and MPLs. Several interesting observations and detailed analyses about these graphs are discussed as follows. We note that ϵ-spec$=$ 1,000 for a query ET is like missing 7.5 updates while scanning 500 data items.

First of all, both 2PLDC and ODC improve their throughputs substantially as ϵ-spec increases. This confirms that ESR indeed helps to improve system concurrency as evidenced by the improvement in the throughputs of UETs in Figure 3(a) and QETs in Figure 3(b).

Secondly, without ESR (i.e., ϵ-spec=0), the system throughput quickly declines as MPL increases. However, with ϵ-spec > 0, the system throughput continues to improve even as MPL increases. The peak of throughput shifts into a higher MPL as ϵ-spec becomes larger. For example, in Figure 3(a), the throughput of 2PLDC starts to decrease at MPL=25 for ϵ-spec=0, while it does not until MPL=75 for ϵ-spec=3,000. The above two observations match our intuition as ESR alleviates the data-contention and it allows a larger number of transactions to progress.

Thirdly, with 10 resource units, 2PLDC performs better than ODC for ϵ-spec=0. With ϵ-spec > 0, however, the difference between 2PLDC and ODC becomes less distinguishable. In order to understand this phenomenon, we inspect the disk utilizations in Figure 4 and abort/commit ratio in Figure 5.

In Figure 4, 2PLDC extends the disk usage as ϵ-spec increases, while ODC maintains almost the same in the uppermost overlapped curves for all ϵ-spec. However, for ODC some of the disk utilization is due to wasteful work of aborted transactions. Note that in ODC transactions use almost the same resources regardless of the ϵ-spec, due to the immediate grant of accesses. However, in 2PLDC the decrease of data contention due to the increase of ϵ-spec causes less blocking and results in less transactions being suspended at the scheduling queue. As a result, more transactions issue disk requests and disk utilization increases as ϵ-spec increases. If we separate the disk utilization of committed transactions from the total disk utilization observed in Figure 4 for ODC, the useful utilization of ODC should be comparable to that of 2PLDC. Table 2 summarizes the disk utilization extracted from MPL=100. Notice that in 2PLDC the observed total utilization is very close to the useful disk utilization because the abort/commit ratio of 2PLDC is very small (see Figure 5). From Table 2, the useful disk utilization of ODC, from ϵ-spec=3,000, becomes comparable to the observed total disk utilization of 2PLDC. Thus, with limited resources and higher ϵ-spec, ODC and 2PLDC performs comparably.

	2PLDC	**ODC**	
ϵ-spec	**Observed**	**Observed**	**Useful**
0	0.31	0.90	0.49
1,000	0.60	0.90	0.64
3,000	0.84	0.90	0.83
5,000	0.90	0.90	0.90
1,000,000	0.90	0.90	0.90

Table 2: Disk Utilization

Another important issue related to the third observation is that a smaller ϵ-spec affects the two algorithms differently. In Figure 3(a), ODC outperforms 2PLDC with ϵ-spec=1,000 when MPL ≥ 100. Figure 6(a) plots the rate of UET's performance improvement in throughput over the case of ϵ-spec=0 for the cases of MPL=50, 100 and 150. For example, the throughput of ODC at

MPL=100 gains 2.35 times from ϵ-spec=0 to ϵ-spec=1,000, whereas 2PLDC improves only 1.37 times. At MPL=150, the rate of improvement in ODC is even larger (5.5 times), which shows that ODC gains an even larger improvement with a higher MPL. The similar property holds for QET as well.

This higher rate of improvement in ODC can be explained as follows. Upon reaching its ϵ-spec limitation, 2PLDC starts to build up a wait-for chain due to blocking. And its performance quickly reaches saturation because all suspended ETs hold locks and eventually result in more conflicts. In contrast, ODC examines conflicts only during an instantaneous moment of the validation stage. Thus, it captures comparably smaller set of interferences among active ETs due to immediate grant of accesses without blocking. It thus enables more ETs, even with a small ϵ-spec, to complete and leave the system.

Finally, let us go back to Figure 3(c) and 3(d) where the response times of UETs and QETs are shown. The response times of UETs are generally shorter in ODC than in 2PLDC (see Figure 3(c)), while the reverse is true for QETs (see Figure 3(d)). In 2PLDC, UETs progress slowly once blocked because they must wait until lengthy QETs complete, resulting in larger response times. On the other hand, in ODC, QET is more likely to be aborted and aborting a QET is costly because it takes a long time to re-execute, while UETs can be re-executed quickly. Thus, the response times of QETs tend to be longer in ODC.

4.2 Experiments with Infinite Resources

In this section, we use an infinite number of CPU and disks and examine the performance impact of data contention. Figure 7(a)–(d) show the throughputs and response times of UETs and QETs.

As in the limited-resource case, the throughputs of both 2PLDC and ODC improve substantially as ϵ-spec increases in Figures 7(a) and (b). However, in contrast to Figures 3(a) and (b), ODC outperforms 2PLDC for almost all the ϵ-spec, including ϵ-spec=0. That ODC is better than 2PLDC with infinite resources and ϵ-spec=0 conforms with previous studies of 2PLCC and OCC algorithms [1, 4]. This is different from the observation that 2PLDC is slightly better than ODC with the resource-limited environment in the previous section.

Figure 6(b) plots the rate of UET's performance improvement versus ϵ-specs using throughput extracted from MPL=50, 100 and 150. Although the rate of improvement of 2PLDC is significantly better than that of ODC in Figure 6(b), 2PLDC begins at a much lower throughput level for the case of ϵ-spec=0. In fact, the throughput of 2PLDC never catches up with that of ODC under any

ϵ-spec.

4.3 Comparisons of Accumulated Inconsistency

In this section, we characterize how the two algorithms differ in the amount of inconsistency in their results by comparing the distributions of imported inconsistency of committed QETs by 2PLDC and ODC.

Figure 8(a)–(c) show the histograms[4] (normalized frequency of sampled values) drawn at MPL=150 in resource-limited setting. There is a significant difference between the two algorithms. For a small ϵ-spec (see Figure 8(a)), a significant portion of committed QETs has a larger accumulated inconsistency in 2PLDC than in ODC. For instance, the three highest buckets of 2PLDC hold 32% of the total samples in Figure 8(a). If we increase ϵ-spec, the inconsistency histogram become less skewed toward the high end and more evenly distributed (see Figures 8(b) and (c)).

This phenomenon is closely related to the implementation of the two algorithms. In 2PLDC, an ET is suspended when ESR extension fails, and it is blocked and waits until it succeeds. While blocked, the accessed data items of a query ET can continue to be updated and result in more inconsistency to accumulate for the waiting QET. However, in ODC an ET is marked for abort whenever ESR extension fails. Therefore, most committed QETs end up with higher inconsistency in 2PLDC than in ODC.

The spikes in the higher buckets in Figures 8(a)–(c) for 2PLDC diminish in magnitude when MPL reduces (see Figure 8(d) where MPL is 100). This is because lower MPL will result in less data contention and smaller inconsistency for QETs.

5 Summary

In this paper, we developed a comprehensive simulation program to evaluate and compare performance gains by two-phase locking divergence control (2PLDC) and optimistic validation divergence control (ODC) for transaction processing based on epsilon serializability (ESR). Our simulation extends Agrawal et al's study [1] on the performance of concurrency control methods, whose results are used for validation of our simulation program. We investigated a wide range of workload and system parameter settings and found the following significant results.

[4] The first bucket refers to zero accumulated inconsistency. The last bucket means maximum inconsistency. Other buckets divide ϵ-spec into even ranges.

First, ESR extends classic serializability by allowing a bounded amount of inconsistency (ϵ-spec) in each transaction. Therefore, we expect the system concurrency level to rise as ϵ-spec increases. Our simulation results confirm that both divergence control methods (2PLDC and ODC) allow more effective concurrency than traditional concurrency control for serializability, under a mixed load of small updates and long running queries. Both 2PLDC and ODC provide substantial performance improvement (i.e. throughput and response time) even with a small ϵ-spec. More specifically, they achieve better peak performance at higher MPL as ϵ-spec is raised. Thrashing points also shift to higher MPL. This is true under both resource limited and unlimited environments. This result amplifies and complements previous work on timestamp-based divergence control [9].

Second, we compared the performance of 2PLDC and ODC beyond the confirmation of [9]'s results when ϵ-spec=0. On the other end of spectrum, since sufficiently large ϵ-spec values allow free access (without concurrency control) to a great majority of transactions, it matters little which divergence control method is used. For moderate to small values of ϵ-spec, ODC performance is more sensitive to ϵ-spec changes than 2PLDC. Furthermore, as ϵ-spec grows ODC's concurrency improvement rate is faster. For experiments focused on data contention (using infinite hardware resources), ODC allows better throughput and response time for both UETs and QETs; this is particularly obvious at high MPL.

Finally, the simulation program is useful in guiding our ongoing research. For example, we have implemented 2PLDC on Transarc Encina [12], a commercial transaction monitor. The performance measurements of the implementation, using TPC benchmarks, have been limited by hardware resources (less than 20 MPL on a SUN IPX). The simulation shows that by increasing data contention we should be able to measure some interesting results on a SUN Sparc10 server (about 3 to 5 times faster).

References

[1] R. Agrawal, M. J. Carey, and M. Livny. Concurrency control performance modeling: Alternatives and implications. *ACM Trans. on Database Systems,* 12(4):609-654, Dec. 1987.

[2] R. Alonso, D. Barbara, and H. Garcia-Molina. Data Caching issues in an informational retrieval systems. *ACM Trans. on Database Systems,* 15(3):359-384, Sept. 1990.

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

[4] M. J. Carey, and M. R. Stonebraker. The performance of concurrency control algorithms for database management systems. In *Proc. of the 10th Int. Conf. on Very Large Data Bases,* pages 107-118, Aug. 1984.

[5] A. Dan, D. M. Dias, and P. S. Yu. The effect of skewed data access on buffer hits and data contention in a data sharing environment. In *Proc. of the 16th Int. Conf. on Very Large Data Bases,* pages 419-431, 1990.

[6] A. Dan, and D. Towsley. An approximate analysis of the LRU and FIFO buffer replacement schemes. In *Proc. of ACM SIGMETRICS,* May 1990.

[7] A. Dan, P. S. Yu, and J.-Y. Chung. Characterization of database access skew in a transaction processing environment. In *Proc. of Int. Conf. on Data Engineering,* pages 134-143, 1993.

[8] P. Franaszek and J. T. Robinson. Limitations of concurrency in transaction processing. *ACM Trans. on Database Systems,* 10(1):1-28, Mar. 1985.

[9] M. Kamath, and K. Ramamritham. Performance characteristics of epsilon serializability with hierarchical inconsistency bounds. *Proc. of Int. Conf. on Data Engineering,* 1993.

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

[11] C. Pu. Generalized transaction processing with epsilon-serializability. In *Proc. of 1991 Int. Workshop on High Performance Transaction Systems,* 1991.

[12] C. Pu and S .W. Chen. ACID Properties need fast relief: Relaxing consistency using epsilon serializability. In *Proc. of 5th Int. Workshop on High Performance Transaction Systems,* 1993.

[13] C. Pu, W. Hseush, G. E. Kaiser, K.-L. Wu and P. S. Yu. Distributed Divergence Control for Epsilon Serializability. In *13th Int. Conf. on Distributed Computing Systems,* pages 449-456, May 1993.

[14] C. Pu and A. Leff. Execution autonomy in distributed transaction processing. In *Proc. of the 2nd Int. Workshop on Research Issues in Data Engineering: Transaction and Query Processing,* pages 2-11, 1992.

[15] K. Ramamritham and C. Pu. A formal characterization of epsilon serializability. *IEEE Trans. on Knowledge and Data Engineering.* to appear.

[16] H. Schwetman. CSIM Reference Manual (Revision 16). Microelectronic and Computer Technology Corporation, May 1992.

[17] H. Schwetman. CSIM Users' Guide (Revision 2). Microelectronic and Computer Technology Corporation, July 1992.

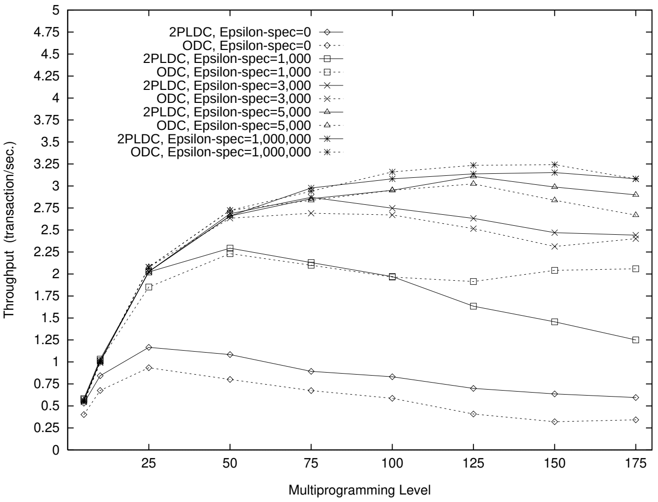
[18] K.-L. Wu, P. S. Yu, and C. Pu. Divergence control for epsilon-serializability. In *Proc. of 8th Int. Conf. on Data Engineering,* pages 506-515, Feb. 1992.

[19] K.-L. Wu, P. S. Yu, and J. Z. Teng. Performance Comparison of Thrashing Control Policies for Concurrent Mergesorts with Parallel Prefetching. In *Proc. of ACM SIGMETRICS,* pages 171-182, 1993.

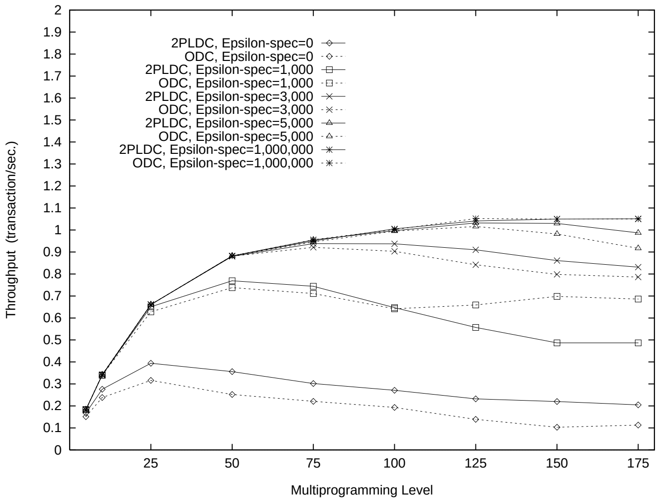
[20] P. S. Yu and D. M. Dias. Performance analysis of concurrency control using locking with deferred blocking. *IEEE Trans. on Software Engineering,* 19(10):982-996, Oct. 1993.

[21] P. S. Yu and D. M. Dias. Analysis of hybrid concurrency control schemes for a high data contention environment. *IEEE Trans. on Software Engineering,* 18(2):118-129, Feb. 1992.

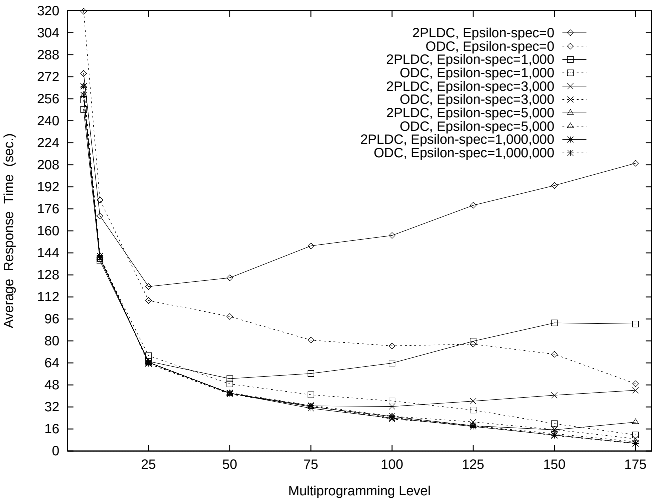
[22] P. S. Yu, D. M. Dias, and S. S Lavenberg. On the analytical modeling of database concurrency control. *Journal of the ACM,* 40(4):831-872, Sept. 1993.



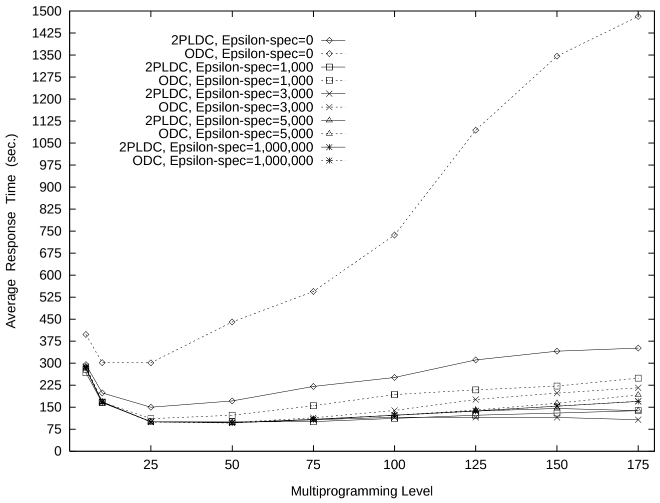
(a) Throughput for UETs



(b) Throughput for QETs



(c) Response Time for UETs



(d) Response Time for QETs

Figure 3: Performance of 2PLDC and ODC with 10 Resource Units

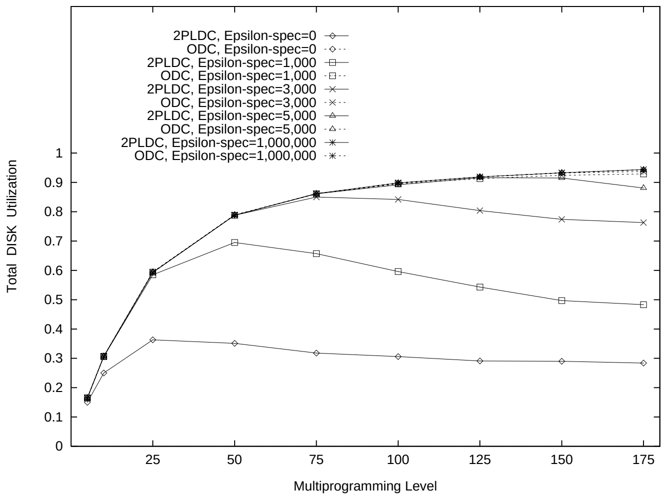


Figure 4: Total Disk Utilization

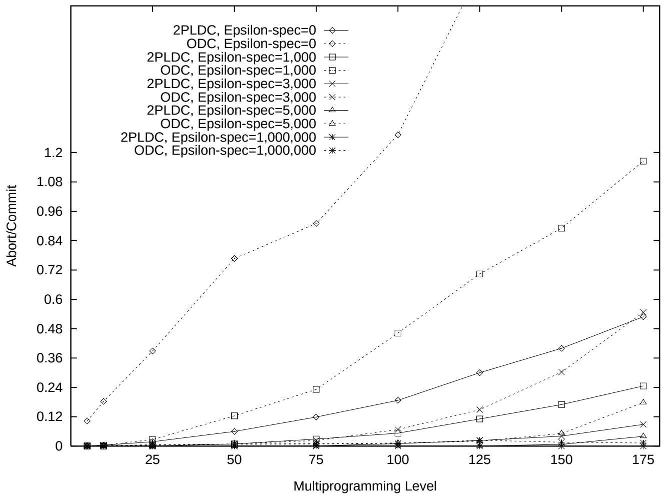
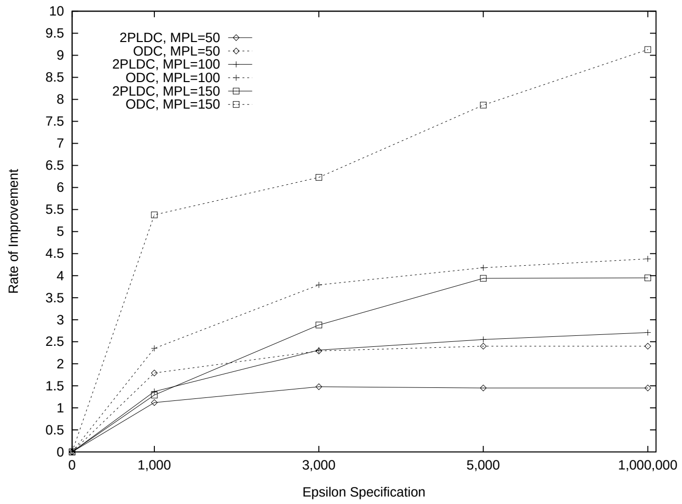
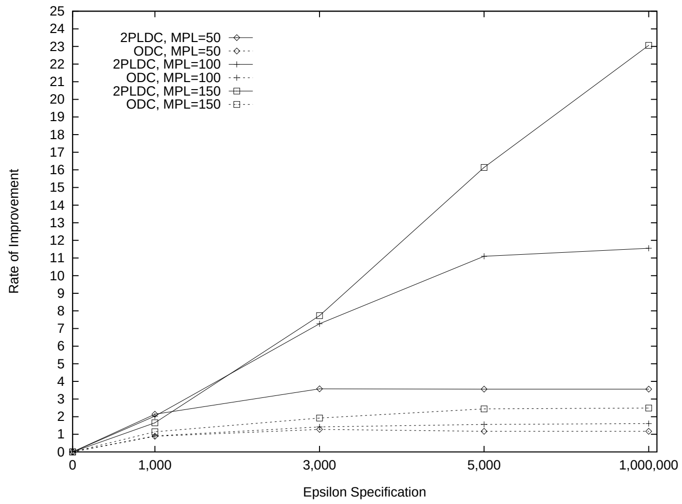


Figure 5: Abort/Commit

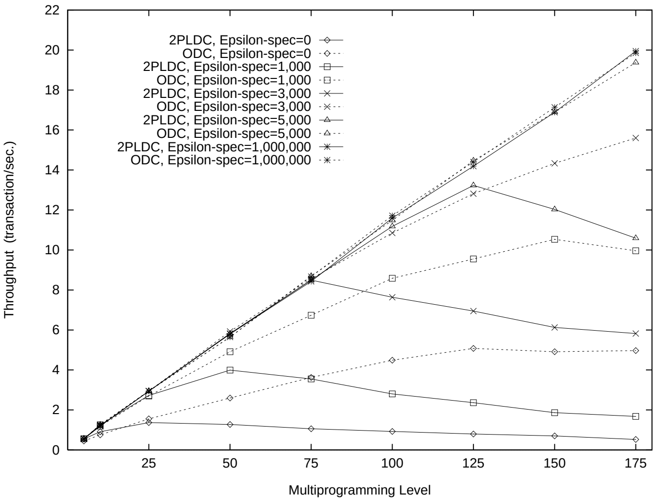


(a) 10 Resource Units

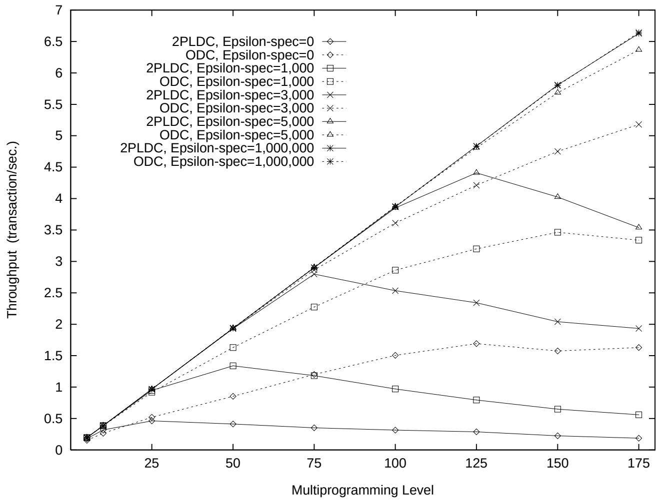


(b) ∞ Resources

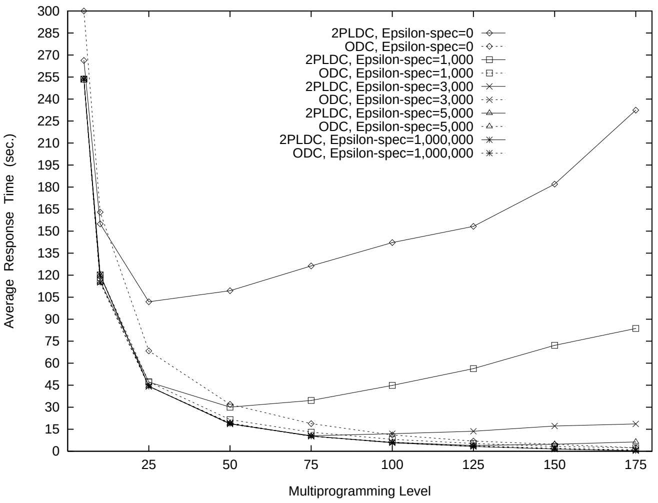
Figure 6: Rate of Throughput Improvement (UETs)



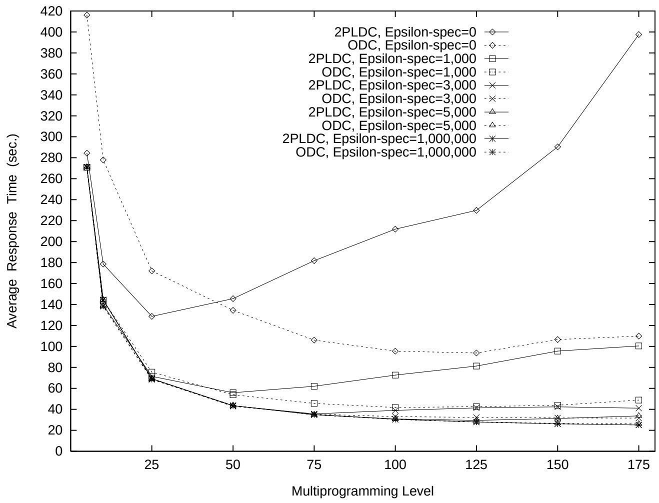
(a) Throughput for UETs



(b) Throughput for QETs

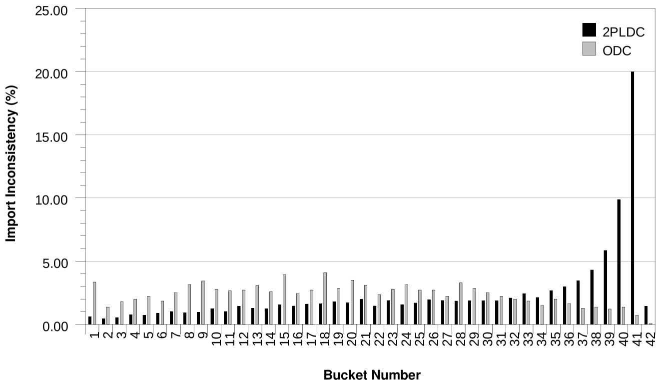


(c) Response Time for UETs

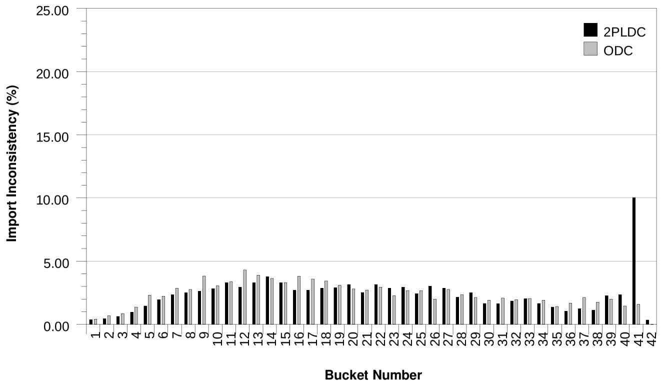


(d) Response Time for QETs

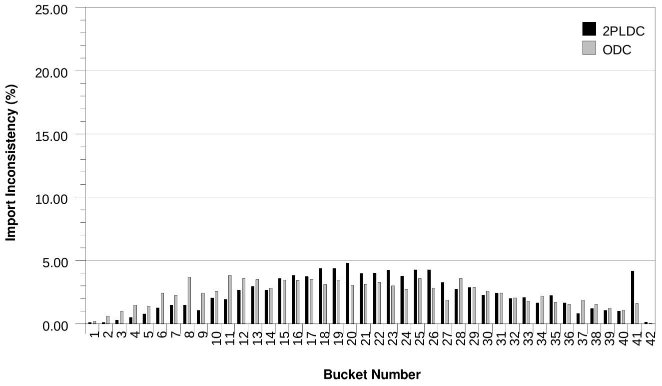
Figure 7: Performance of 2PLDC and ODC with ∞ Resources



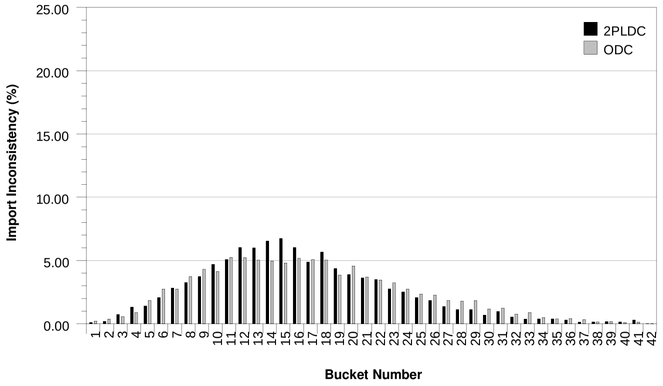
(a) ϵ-spec=1,000; MPL=150



(b) ϵ-spec=3,000; MPL=150



(c) ϵ-spec=5,000; MPL=150



(d) ϵ-spec=5,000; MPL=100

Figure 8: QET Imported Inconsistency