**Response Letter**

We appreciate the reviewers' and the meta-reviewer's comments and their efforts put into the reviews. To address the reviewers' concerns, we have prepared a revision with necessary changes addressing all the comments.

In summary, we have made the following major changes.

First, we fixed some errors and further clarified the presentation, including correcting misuse of some terminologies pointed out by the reviewers and clarification of the concepts of "interactive" and "one-shot" transactions. We also improved the presentation about the motivation and contributions.

Second, we made additional explanations to help readers understand the technical parts. For instance, we used an example to illustrate why 2PC over 2PL should not release locks before replicating the commit log if it wants to preserve linearizability. We provided a more detailed discussion about the timing of lock violation and how it fits in the two-phase commit message flow. We explained why DLV needs not to persist inter-transaction dependency on disks.

Third, we conducted additional experiments to evaluate how different lock violation approaches affect transaction latency, the effects of lock violation on interactive transactions, and the cost of cascade abort.

Forth, we delete several contents to shorten the length of this paper. These contents include some tables, experiment results, and knowledge unrelated to understand this paper.

**Response to the meta reviewer’s comments**

Comments: *Interesting paper on distributed lock management. The revision request is around presentation and missing explanations, like how RMs persist the dependency information, evaluating one-shot vs interactive, and analyzing the cascading aborts.*

Response：We added explanations to make these parts more understandable. We explained why DLV needs not to persist inter-transaction dependency on disks (the last paragraph of Section IV.B). We clarified the definitions of one-shot vs. interactive transactions (Section III.A and Fig. 3) and showed their relations with the DLV's design (Section IV.A). We conducted additional experiments to evaluate the cost of cascading aborts (Section V.G).

**Response to Reviewer 1’s comments:**

*W1. The paper does not give enough explanations to some important concepts and techniques which are necessary for readers to understand the design of DLV.*

Response: We made additional explanations to make the technical parts more understandable. We clarified the definitions of one-shot vs. interactive transactions (Section III.A and Fig. 3). We showed why 2PC over 2PL should not release locks before replicating the commit log to preserve linearizability (Section I, Page 2, 2nd paragraph). We provided more deals of the different DLV designs (Section IV.A). We explained why DLV needs not to persist inter-transaction dependency on disks (the last paragraph of Section IV.B).

*W2. The paper does not discuss and prove how RMs persist dependency information and achieve fault-tolerance.*

Response: Persisting dependency information will introduce additional overhead. DLV chooses not to persist dependency information in stable storage. In DLV, each RM independently maintains its dependency information locally in memory. (If two transactions conflict on an RM, their dependency introduced by the conflict will be recorded only in that RM.) When a node fails, the dependency information on the node can all be discarded. If a transaction’s in value on an RM is greater than 0, we can guarantee that this RM has not entered the uncertain “prepare commit” state. Upon a failure, this un-prepared RM can simply vote to rollback the transaction without referring to the dependency information. Otherwise, if the RM is already prepared, we do not need the in value to proceed. Therefore, it is safe to discard the dependency information when a failure occurs. We have added this part of explanation to Section IV.B.

*W3. The experiment part does not show transaction latency, which could be interesting for readers, since DLV is mainly used to shorten the lock holding time.*

Response: In fact, DLV does not necessarily benefit transaction latency. Although it has shortened the lock holding time, a transaction still needs to hold until all its preceding transactions end. This final wait phase is necessary to ensure transactions' correctness. We have added a set of experiments to evaluate the effects of DLV on transaction latency (Section V.H). It shows that latency is more affected by the system's saturation level rather than by lock violation.

*D1. The first main contribution of this paper says that “CLV does not necessarily help in enhancing the performance of GDDB”. But (a) the paper has only a little explanation about what CLV is in the Related Work, which is not enough to be a main contribution; (b) It proposed DLV as a variation of CLV, but not clearly explain the differences between them; (c) it has no experiments to support the claim.*

Response: We added more discussion about CLV and how to apply it to distributed databases (Section II.B). We also clarified the description of our contributions accordingly(Section I).

*D2. The definitions of one-shot and interactive transactions are not clear. The paper does not explain (1) why they summarize these two types of transactions, (2) why their message flows are different and what features the transaction has to make the difference, (3) why it is necessary to do so. And there are no experiments comparing*

Response: In the previous version, interactive and one-shot transactions were not clearly defined. In the revised version (Section III.A), we used a different pair of terminology to classify the transactions and explained why they are related to our design choices.

Basically, if a transaction knows that it will not violate serializability before issuing the prepare message, it is a DP (deterministic prepared) transaction. Otherwise, it is an NDP (non-deterministic prepared) transaction. In a DP transaction, if an RM receives a prepare message, it knows that other RMs of this transaction will all be prepared sooner or later, as long as there is no failure. In practice, most interactive transactions can be categorized as DP transactions, but not all. Fig. 3 in the revised version illustrates the difference. The distinction between DP and NDP will affect how DLV conducts lock violation and dependency tracing when an RM receives a prepare message. (In principle, it only affects the designs of DLV1 and DLV1x.)

We have conducted an additional evaluation on interactive transactions (DP transactions) (Section V.F).

*D3. The message flow in Figure 8 is not so clear. It is better to mark which phase of 2PC the message is for. And the paper does not explain how the serializability test message sending from RMs to TM comes.*

Response: We had marked the message type in the figure. We have also added some explanation on how the serializability test is issued and what it is used for (Section IV.A and Fig. 8).

*D4. The paper let RM nodes maintain the dependency information, but does not explain how it is persisted to achieve fault tolerance.*

Response: See our response to W2.

**Response to Reviewer 2’s comments:**

*W1. The only minor concern I have is that the techniques are fairly straightforward extensions of existing lock violation ideas from centralized databases. But I believe the paper sufficiently compensates for this by thoroughly exploring these ideas in a new context. I enjoyed reading the technical discussion and the experiments.*

Response: This is quite right. We applied the old idea of lock violation to geo-replicated distributed database. Nevertheless, we presented and evaluated different lock violation approaches on GPDB, and some results are not trivial. We hope our results and insights can be helpful to future developers of distributed database systems.

*W2. I would like to see a discussion how a GDDB or DB admin would select the best DLV option for their application. How can this be decided automatically? From the experiments, it looks like DLV1x is the winner. Does this generalize to other applications?*

Response: The evaluations show that early lock violation fails to scale when the degree of concurrency is high. It also performs worse in handling workload with high abort rates. However, it can achieve good performance when the system is not saturated. In contrast, late lock violation is more adaptive to various workloads and scales well.

Automatically deciding DLV scheme to use sounds like a good idea. We believe such automation is possible, while it requires a lot of work. This can be an interesting topic for future research.

*D1: There really is not much else to be said, except for a few typos. Here are some examples: Sec II.B "This has been illustrated in Fig. 1." should reference Fig. 2 instead. Sec IV.A "...unless it performs a serializability test as DLV1 does." should refer to DLV1x. Sec V.C "400,00 rows" should probably be 400,000 rows.*

Response: Thanks for pointing out the errors. We have fixed them. We also went through the paper and fixed a few other errors and typos.

**Response to Reviewer 3’s comments:**

*W1. I would appreciate more analysis on cascading aborts.*

Response: We have added an experimental study on cascading aborts (Section V.G). Cascade abort seldom occurs for late lock violation, as it violates lock after the transaction finishes its work successfully. Early lock violation is more vulnerable to cascade abort. Nevertheless, our experiment results showed that, even in early lock violation, cascade aborts constituted only a small proportion of the actual aborts. We believe that in general cases, such as TPC-C, cascade aborts will not be a serious threat to the performance.

*W2. There is no evaluation of one-shot vs. interactive.*

Response: In the revision, we refined our definitions of one-shot and interactive transactions (Section III.A). We also included an additional experimental study on interactive transactions (Section V.F).

*D1. I don't quite understand the "one-shot" execution in this paper. My understanding of "one-shot" is that the user provides all the input parameters at the beginning of the transaction so there is no further interaction between the database and the user before commit/abort. The "one-shot" execution in this paper seems different: the TM immediately starts the 2PC process without reading data from RMs beforehand. This does not seem to work if the transaction contains complex logic; for example, the data to write depends on some values read. Please clarify the use case the "one-shot" execution here.*

Response: Thanks for pointing this out. There was some misuse of one-shot transactions and interactive transactions in the previous version.

In the revised version (Section III.A), we used a different pair of terminology to classify the transactions and explained why they are related to our design choices. Basically, if a transaction knows that it will not violate serializability before issuing the prepare message, it is a DP (deterministic prepared) transaction. Otherwise, it is an NDP (non-deterministic prepared) transaction. In a DP transaction, if an RM receives a prepare message, it knows that other RMs of this transaction will all be prepared sooner or later, as long as there is no failure. In practice, most interactive transactions can be categorized as DP transactions, but not all. Fig. 3 in the revised version illustrates the difference. The distinction between DP and NDP will affect how DLV conducts lock violation and dependency tracing when an RM receives a prepare message.

*D2. Related to D1, in the evaluation section, are the transactions modeled as "one-shot" or "interactive"? Would these two models lead to different design decisions?*

Response: We included an additional experimental study on interactive transactions (DP transactions) (Section V.F). We showed that DLV1 and DLV1x demonstrate similar performance in interactive transactions, as they tend to violate locks at the same stage. This is quite different from the results in NDP transactions.

*D3. Please consider flipping the order of Figures 9 and 10. They appear in the opposite order in paper.*

Response: We corrected the order in the revised paper.

*D4. Please report some analysis on cascading aborts. For example, what percentage of aborts are caused by cascading aborts? What is the main cause of cascading aborts? (e.g., detected deadlocks or aborts due to WaitDie) Would different variants of DLV lead to different numbers of cascading aborts?*

Response: Please refer to our response to W1. As our experimental results show, cascade aborts are more likely to be triggered by active aborts, e.g., aborts caused by accessing non-existent data.

*D5. This last comment is more like a question rather than an evaluation comment: In Fig 2, why does the locking critical path end after RM finishes logging? I thought it could end when the RM receives the commit request from the TM. Because at that point, the transaction is already destined to commit.*

Response: We added an example in Section I (Page 2, 2nd paragraph) to explain why it is necessary to preserve locks until the commit log is replicated. We showed that 2PC over 2PL might violate linearizability if it releases locks before replicating the commit log. While serializability and recoverability can in theory be preserved even if we release lock earlier, we may not be willing to sacrifice linearizability, as it is often desired by applications .