Response to Reviews for Paper 723 "*Improving Optimistic Concurrency Control Through Transaction Batching and Operation Reordering*" in Research Track, March 2018

Bailu Ding, Lucja Kot, Johannes Gehrke

We would like to thank the reviewers and the meta-reviewer for their insightful comments and suggestions. We hope that we addressed all the issues that were pointed out. Below is a summary of the changes we made in response to these comments. Changes are marked in blue in the response and the revised submission.

1 Changes based on meta-reviewer comments

(R1) *Expand the experimental evaluation to include a comparison with other high-performance OLTP engines that use OCC.*

We integrated our techniques into Cicada, an open-source high-performance OLTP engine based on OCC, and we compared its performance with the state-of-the-art in-memory OCC-based OLTP engines, such as Cicada, Silo, TicToc, and ERMIA. Our techniques improve throughput by up to 2.2x and reduce the tail latency by up to 71% compared with the next best among them on the write-intensive YCSB benchmark.

We would also like to note that we used exactly the same parameter settings as in the original Cicada paper. The new results are described in Section 5.6.

(R2) *Fix typos, unclear parts, and expand discussion per the reviewers requests.*

We fixed all the them; please see responses to comments from individual reviewers for details.

2 Changes based on individual reviewers' comments

In this section we explain how we addressed the comments of individual reviewers in case the comments are not directly subsumed by a meta-reviewer comment. We have addressed comments on clarifications of the writing, and omitted similar comments that can be addressed by previous responses.

2.1 Reviewer 1

(R1.1) *Add an experiment to show the performance of the proposed techniques in systems where the clients do not send their next transaction until their previous one is finished.*

We integrated our techniques to Cicada, where each thread processes a transaction synchronously. Please see Comment R1 for details.

(R1.2) *Add a more detailed description regarding the DBMS-X experiment.*

Our prototype issues transactions to DBMS-X via JDBC, where it can submit transaction statements individually or in batch. Either way, the connection will block until all the submitted transactions get processed, i.e., commit or abort. As DBMS-X receives a batch of transaction statements from a connection,

it can execute them concurrently. Batching transaction statements reduces the overhead of communication and can increase the concurrency level of DBMS-X as it receives more active transactions.

We added a more detailed description of why batching helps DBMS-X in Section 5.7.

(R1.3) *Improve the legends of the figures.*

We updated the legends to replace unnecessary abbreviations, e.g., *bch* → *batch*.

(R1.4) *Add an experiment to demonstrate the generality of the sort-based algorithm by using more policies or benchmarks.*

While the full generality of the sort-based algorithm will be hard to show, we extended the algorithm with a thread-aware policy for an important class of multi-threaded, decentralized OLTP architecture, where each transaction is processed independently and synchronously by a dedicated thread. We thus believe that our sort-based approach is rather general, and that we have shown a wide variety of both scenarios and different types of systems where this approach has merit.

We implemented the policy on top of Cicada; please see Section 5.6 for the evaluation details.

2.2 Reviewer 2

(R2.1) *Add a discussion which explains what constraints on the system make the system best suited for integration with an in-memory versioned key-value store and what would need to change to be integrated with other stores.*

We removed this statement from the introduction. In our experiments, we integrated our techniques into two systems that are not based on key-value stores, i.e., Cicada (supporting hash index and B+ tree index) and DBMS-X (using the BW-tree). The result shows that a variety of storage and transaction processing architectures can benefit from our techniques.

(R2.2) *There are many moving parts in the experiments so it is difficult to picture the tradeoffs holistically. I suggest that either the results are represented in a formula or represented in a table showing the tradeoffs.*

We summarized the trade-off of different parameters and policies in Table 1 Section 5. In addition, our implementations both on DBMS-X and on the open-source system Cicada show that our approach can be relatively easy to incorporate into other systems to improve their performance.

2.3 Reviewer 3

(R3.1) *The papers contains a bag of ideas/optimizations, arguably unrelated, based on known techniques, so the overall contributions and novelty is limited.*

We believe that our contribution is a rather general framework for batching and associated policy-based reordering of transactions for OLTP systems. As we have shown, our sort-based framework can be customized for different transaction processing performance objectives and for different system architectures. Thus, we believe that our contributions are novel, and that they will have practical impact.

(R3.2) *Although the evaluation is comprehensive and detailed, but the authors only presented a micro benchmark, a self-comparison without considering other state-of-the-art approaches. Here are few important related CCs (related work discussion can also take into these approaches as well)*

We integrated our techniques into an open-source OLTP system (Cicada), and we compared its performance with the state-of-the-art in-memory OLTP systems such as Cicada, Silo, TicToc, and ERMIA. We believe that this comparison is quite comprehensive and provides a fair comparison to the current state-of-the-art. Please see Comment R1 for more details.

(R3.3) *The authors provide a black box comparison of their approach in DBMS-X, but the basis of the comparison is unclear (also not sure what does "Good Throughput/Transaction" mean). Although the effort is appreciated, it would be much better to compare with relevant, disclosed existing algorithm, or at the very least, the concurrency of the model of DBMS-X is carefully explained. A black box graph adds no value.*

"Good Throughput" means the number of committed transactions per second, since some literature refers to "Throughput" as the number of transactions processed per second, no matter they commit or abort. We changed to term to "throughput" and clarified its semantics in Section 5.1.

We cannot disclose the name of DBMS-X for confidentiality reasons, but it uses a very specific OCC protocol. However, our techniques are not at all customized for its protocol. We also added a whole new experiment to compare with other state-of-the-art OCC-based OLTP systems for a comparison beyond DBMS-X. Please see Comment R1 for details.

(R3.4) *It is unclear why the authors choose to have 300 active transactions, which would imply the need for having 300 physical threads. I further suppose, the authors considering in-memory implementation given all OLTP DB can fit entirely in memory today. If in fact, the number of active transactions is larger than threads, then there will be many context switches, and frankly, the whole setting would be questionable, which could also explain why the overall throughput is never exceeded half-million transactions/second.*

As described in Section 5.1, our prototype for the micro-benchmark experiment is implemented with asynchronous transaction processing architecture. Each thread multiplexes multiple transactions simultaneously to mask the latency of transaction workflow execution, IO accesses, and network communication. Thus, the number of threads can be much less than the number of active transactions. This architecture has loosely coupled components and can scale storage and compute independently, which is especially applicable for elastic transaction processing in the cloud.

We also integrated our techniques to Cicada, where each thread processes a transaction synchronously and independently. In this integration, each thread is pinned to a dedicated physical CPU core and the number of active transactions never exceeds the number of cores. We achieved close to 2 millions transactions per second under highly skewed, write-intensive workload, which is up to 2.2x of the throughput compared with the state-of-the-art OCC-based OLTP systems.

(R3.5) *In some graphs, x-axis label is cut off.*

Thank you for catching this. We fixed the truncated labels and ticks.