2018 IEEE 34th International Conference on Data Engineering

On the Design of Adaptive and Speculative Concurrency Control in Distributed Databases

Qian Lin
National University of Singapore
linqian@comp.nus.edu.sg

Gang Chen
Zhejiang University
cg@cs.zju.edu.cn

Meihui Zhang
Beijing Institute of Technology
meihui_zhang@bit.edu.cn

***Abstract*—Efficient online transaction processing is key to many database applications, and existing concurrency control protocols perform remarkably well under specific workloads or access patterns that they have been designed for. However, they often do not scale well when the workload is dynamic. To tackle the challenge of dynamic workloads, we propose an Adaptive and Speculative Optimistic Concurrency Control (ASOCC) protocol for effective transaction processing. Based on real-time monitoring of data access frequency, ASOCC adaptively embeds 2PL into the OCC scheme to facilitate superior contention resolution with reduced transaction aborts. Further, ASOCC dynamically inspects the correlation of data accesses and exploits such information to perform speculative transaction restart to save CPU cycles wasted on the processing of transactions that are destined to abort.**

***Keywords*-OLTP; distributed database; concurrency control; OCC; 2PL; speculation**

I. Introduction

With considerable efforts devoted to the research of efficient concurrency control and high performance transaction management, it is not surprising that existing concurrency control protocols for online transaction processing (OLTP) exhibit remarkable performance under specific workloads which they have been designed for. For example, optimistic concurrency control (OCC) achieves high parallelism for low-contention workloads; two-phase locking (2PL) is effective for resolving conflicts under highly skewed contended workload; and multiversion concurrency control such as snapshot isolation is suitable for read-heavy workloads with relaxed serializability requirement [1]. However, the design of existing concurrency control protocols lack sufficient consideration of workload diversity at runtime. Consequently, concurrency control schemes do not adapt well to dynamic workloads and result in low throughput. To address such an issue, it is necessary to take into account insights from the workload at runtime, and exploit the runtime dynamics to tune the concurrency scheme autonomously.

For in-memory OLTP, although state-of-the-art OCC schemes [2, 3, 4] offer superiority in multi-core systems, such performance is difficult to achieve in a distributed cluster where distributed transactions are involved [5]. Distributed transactions are not only expensive, but also vulnerable to excessive transaction aborts [6]. Therefore, it is important to reduce transaction aborts that can be avoided while retaining high parallelism of transaction processing. In this paper, we propose a variant of the OCC protocol, called Adaptive and Speculative OCC (ASOCC), for distributed in-memory transaction management. Inspired by memory management techniques, the proposed ASOCC protocol classifies data based on the frequency and correlation of data accesses into cold, hot and warm categories [7], and dynamically applies different strategies of concurrency control according to the data access patterns. ASOCC adopts the original OCC scheme for cold data accesses, while it adaptively uses 2PL for hot data accesses as a means to reduce transaction abort. Specifically, the 2PL variant adopted in ASOCC is the strong strict 2PL (SS2PL) with the Wait-Die policy for deadlock prevention [1]. Further, ASOCC speculatively performs early restart of transactions against conflicting warm data accesses so that CPU cycles for the transactions that are destined to abort can be saved.

II. The ASOCC Protocol

A. Preliminaries

OCC and 2PL are the main building blocks of the proposing ASOCC protocol. As an optimistic scheme, OCC presumes conflicts of data accesses in the workload to be rare. It allows concurrent transactions to run in their individual private space and postpones conflict resolution until transactions are about to commit. As a consequence, the concurrency control overhead incurred by OCC is marginal, but its conflict resolution is expensive since excessive aborts will waste tremendous CPU cycles spent on unsuccessful trials of transaction execution. In contrast, 2PL is a pessimistic concurrency control since it blocks a transaction until related transactions with conflicting data accesses are done. For every transaction, the 2PL scheme restricts lock acquisition in a *growing phase* and lock releasing in a *shrinking phase*. Using 2PL, conflicting data accesses can be detected early along transaction execution, and the conflicting transactions are blocked rather than immediate abort unless deadlock occurs. Therefore, 2PL is more suitable than OCC for workload with high contention since 2PL would incur lower abort rate.

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00154

IEEE computer society

B. Protocol Design

The ASOCC protocol is designed to switch between different concurrency control strategies with respect to the data access pattern, e.g., frequency and correlation of data accesses. Fig. 1 illustrates the workflow of ASOCC. Essentially, data are classified into three categories: cold, hot, and warm. While the classification for cold and hot data is based on the frequency of data accesses [7], the classification of warm data is based on the correlation of data accesses. Access to cold data simply follows the OCC scheme that consists of the read, validation and write phases [1], and access to hot data entails the acquisition of the corresponding data lock in the read phase and release of the lock in the write phase. In other words, the SS2PL scheme is embedded into the OCC scheme for hot data accesses. Consequently, since conflict-free accesses to hot data can be guaranteed by the locks, the validation phase in this case becomes trivial and thus can be bypassed. Access to warm data involves an immediate, speculative validation and may subsequently trigger transaction restart (without releasing the acquired locks) if the validation detects any conflict. Warm data is profiled by a probability P_{spec} which is proportional to the related data access conflicts being observed.

In ASOCC, transaction abort or restart may occur in the following three cases (as illustrated in Fig. 1):

- *Abort due to Failed Validation ($Abort_{occ}$)*: Update of a cold or warm data failing the validation will result in transaction abort.
- *Abort due to Failed Lock Acquisition ($Abort_{lock}$)*: Lock acquisition for hot data access being rejected will result in transaction abort.
- *Speculative Restart ($Restart_{spec}$)*: Transaction may restart in the read phase with warm data accesses. Such transaction restart derives from the speculation strategy.

Note that abort due to $Abort_{occ}$ can retain all the acquired locks and restart another trial of transaction execution, whereas abort due to $Abort_{lock}$ has to release all the acquired locks before the transaction restarts so that deadlock can be avoided.

C. Gain Analysis

It has been a known issue that OCC suffers from excessive transaction aborts under high-contention workload [8]. According to the OCC scheme, transaction abort happens in the validation phase, where the entire processing of the transaction has been finished except for the commit. As transaction abort would definitely waste all (or most of) the CPU cycles spent on partial processing of the transaction, it is beneficial to abort transactions that are going to abort as early as possible [9]. This motivates the design of the ASOCC protocol as presented. As a variant of OCC, we reason about the gain of ASOCC by comparing it with OCC. We presume the existence of highly contended data accesses that would potentially lead to considerable transaction aborts.

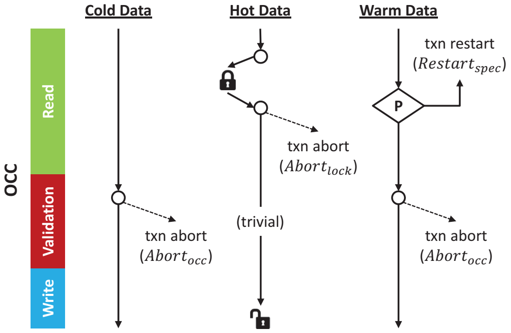


Figure 1. The ASOCC protocol.

- If a transaction is destined to abort, $Abort_{lock}$ is more economic than $Abort_{occ}$ regarding the wasted CPU cycles for partial processing of the transaction. This is because $Abort_{lock}$ can interrupt and restart the transaction execution upon any unresolvable conflict of data access detected. In contrast, although conflicting data access which leads to $Abort_{occ}$ may appear in the middle of transaction execution, the transaction is agnostic to the conflict until it enters the validation phase. Therefore, $Abort_{occ}$ brings about inevitable waste of CPU cycles spent on the invalid processing.
- Similarly, as long as warm data are properly categorized, $Restart_{spec}$ tends to save abundant CPU cycles spent on the transactions that are likely to abort.

Moreover, under low-contention workload, cold data accesses tend to dominate the workload. This makes ASOCC degenerate to OCC and consequently inherit the superiority of transaction processing with high parallelism (comparing with the purely pessimistic 2PL scheme).

III. Speculative Concurrency Control

While the hybrid OCC/2PL scheme employed in ASOCC shares similarity with recent works in literature [10, 11, 12] regarding the design philosophy, ASOCC steps further to take into account speculation for concurrency control. This is motivated by our key observation that correlation of data accesses is another important factor reflecting data access pattern.

A. Correlation between Data Accesses

In many real-world applications, the business logics or user behaviors, more often than not, imply some kind of correlation of data accesses within certain context. We formally define the correlation of data accesses as follows:

Definition 1 (Data Access Correlation). For data records r_1 and r_2, if the access to r_1 accompanies the access to r_2 in the same transaction with high probability, then the access to r_1 is deemed to be correlated with the access to r_2.

Towards improvement of concurrency control, we are only interested in the correlation where one data is hot and the other is cold. This is because if the correlated data are both hot or both cold, their accesses can be well handled by the hybrid OCC/2PL scheme.

Definition 2 (**Warm Data**)**.** For data records r_1 and r_2, r_1 is hot but r_2 is not. If the access to r_2 is correlated with the access to r_1, then r_2 is considered to be warm with r_1.

B. Speculative Transaction Restart

A key observation regarding warm data accesses is that accesses to warm data could be potentially contended. Let data r_1 and r_2 be two data records such that r_2 is warm with r_1. Since r_1 is hot, transaction T_1 accessing r_1 needs to acquire the lock of r_1. If the lock cannot be granted immediately, some other transaction T_2 must be manipulating r_1. Meanwhile, we could infer that T_2 is probably also accessing r_2 due to the data access correlation between r_1 and r_2. After T_1 obtained the lock of r_1, though its exclusive access to r_1 can be guaranteed, conflict of accessing r_2 may exist if T_1 needs to access r_2 but r_2 has been modified by T_2. In this case, T_1 is likely to abort in the validation phase due to its access to the stale r_2. If T_1 is going to abort but can be restarted early, CPU cycles wasted on T_1 due to its abort could be saved.

More specifically, we only consider speculative transaction restart when the access to a warm data precedes the access to the correlated hot data in the same transaction. This is because, if such access order is reversed, the pessimistic concurrency control for hot data access can also “protect” the subsequent warm data access and thus doing speculation becomes unnecessary.

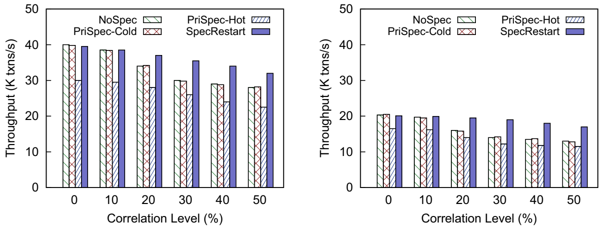
The strategy of speculative transaction restart in the ASOCC protocol is designed as follows. Each transaction speculatively validates warm data accesses if any related lock request has experienced pending, and successful validation entitles the transaction to continue its execution. The arbitration of triggering transaction restart evaluates the conjunctive validations of the accessed warm data that are correlated with the accessing hot data. Transaction restart is triggered if any of the conjunctive validations fails. By doing so, indispensable early restart of transaction processing can be guaranteed by the validation that fails. On the other hand, if the speculative validation, which is performed in the read phase, succeeds (i.e., all the related warm data accesses are still valid), then we have to pay for the redundant validation since warm data accesses additionally have to be validated in the validation phase. To avoid performance penalty due to superfluous validation of warm data accesses, we need to tune the categorization algorithm to prevent from bringing in too many false positives of warm data.

IV. Performance Evaluation

We have implemented ASOCC on top of HoroDB, a research prototype of distributed in-memory database coded in Scala. HoroDB is a partitioned database and adopts the two-phase commit (2PC) protocol [1] to coordinate distributed transactions. For concurrency control schemes, the OCC implementation adopts the backward validation [1], and the 2PL implementation adopts the SS2PL with the Wait-Die policy for deadlock prevention [1]. We experiment using the TPC-C benchmark. In particular, to focus on the study of distributed transactions, only the NewOrder and Payment procedures are used to generate transactional workloads since these two procedures involve remote data accesses to the Customer and Stock tables.

Due to space constraints, we specially evaluate the speculative concurrency control of ASOCC, which can be treated as an add-on optimization for the hybrid OCC/2PL scheme. To this end, we synthetically add correlation of data accesses when generating the workload. The *correlation level* is defined as the conditional probability of occurrence of warm data access given that the corresponding hot data is accessed within the same transaction. We add one correlated data access in each NewOrder transaction, and vary the correlation level from 0% to 50% in different runs. Fig. 2 compares the transaction processing throughput of ASOCC when the speculation strategy is enabled (SpecRestart) and disabled (NoSpec). We additionally implement the priority-based speculative concurrency control [13] and measure its performance as our comparative baseline. In particular, the workloads for running priority-based speculation are crafted with two configurations of priority assignment: One grants priority to certain transactions with cold data accesses only (PriSpec-Cold), and the other grants priority to certain transactions with hot data accesses (PriSpec-Hot).

We can obtain three key insights from the results shown in Fig. 2. First, the speculative concurrency control of ASOCC brings performance gain when data correlation is at the medium level. For example, for 20% and 30% correlation, the throughput of SpecRestart outperforms that of NoSpec. Such performance gain is expected, since the increased contention of warm data accesses tends to cause transaction abort and the speculative transaction early restart can help save the CPU cycles wasted on invalid processing. Second, the speculative concurrency control of ASOCC benefits more efficient processing of distributed transactions. Specially, as shown in Fig. 2.b, SpecRestart shows 35% throughput improvement comparing to NoSpec when the workload contains 10% remote data access with the correlation level being 30%. Such performance improvement is due to the save of those expensive but unnecessary aborts of distributed transactions. In addition, we also observe that ASOCC hardly performs when the workload contains excessive distributed transactions, e.g., attributed to over



(a) 1% remote access

(b) 10% remote access

Figure 2. Effectiveness of speculative concurrency control.

50% remote data accesses. This is because the overhead of 2PC processing in such circumstance dominates the system cost [6] and consequently the benefit brought by speculative concurrency control vanishes. Third, the speculative concurrency control of ASOCC outperforms the priority-based concurrency control. In particular, `PriSpec-Cold` exhibits almost the same performance as `NoSpec`. This is expected since the prioritized transactions with cold data accesses are likely to commit even without speculative execution. On the other hand, `PriSpec-Hot` actually shows the worst performance. This is because the speculative execution of the prioritized transactions with hot data accesses tends to abort, which degrades the system performance due to excessive CPU cycles wasted on the invalid speculative execution.

V. Related Work

Speculative concurrency control for transaction management was initially adopted in real-time database systems. Bestavros et al. [13] proposes to execute multiple "shadow" instances for an urgent transaction to intentionally boost its commit. The number of shadows is configured with respect to the priority of transaction. Such speculative concurrency control exploits aggressive redundant execution in the transaction granule towards quick commit of prioritized transactions, and the strategy is application-oriented since the prioritization always refers to the business logic. Alternatively, ASOCC performs speculation with respect to data access pattern, and the approach is application-agnostic and more fine-grained as in the data record granule.

There are also some works on speculative concurrency control in the data record granule. Johnson et al. [14] adopts 2PL and speculatively passes the locks on hot data directly from transactions to transactions to mitigate the burden of the centralized lock manager. Larson et al. [15] adopts multiversion concurrency control and advocates an optimistic approach to read uncommitted writes of concurrent transactions, taking the risk of cascading aborts. ASOCC differs from these approaches as its speculation strategy is devised based on the runtime statistics of data accesses.

VI. Conclusion

We proposed the ASOCC protocol for adaptive and speculative concurrency control in distributed transaction management. ASOCC requires no prior knowledge of OLTP workload runtime, and can automatically adapt its concurrency control scheme to the dynamics of workload with respect to the real-time monitoring of data accesses. It also benefits reduced cost of transaction abort through the strategy of speculative transaction restart. According to our experimental study, the speculative concurrency control of ASOCC exerts positive effect when the workload contains medium level of hot-warm data correlation and the percentage of distributed transactions is not too high.

Acknowledgments

The work in this paper is supported by the National Research Foundation, Prime Ministers Office, Singapore under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08). The work is also in part supported by the Tencent-NUS Collaborative Research Grant.

References

[1] G. Weikum *et al.*, *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers Inc., 2002.

[2] X. Yu *et al.*, "Tictoc: Time traveling optimistic concurrency control," in *Proc. of SIGMOD*, 2016.

[3] Y. Wu *et al.*, "Transaction healing: Scaling optimistic concurrency control on multicores," in *Proc. of SIGMOD*, 2016.

[4] S. Tu *et al.*, "Speedy transactions in multicore in-memory databases," in *Proc. of SOSP*, 2013.

[5] R. Harding *et al.*, "An evaluation of distributed concurrency control," *PVLDB*, vol. 10, no. 5, pp. 553–564, 2017.

[6] Q. Lin *et al.*, "Towards a non-2pc transaction management in distributed database systems," in *Proc. of SIGMOD*, 2016.

[7] J. J. Levandoski *et al.*, "Identifying hot and cold data in main-memory databases," in *Proc. of ICDE*, 2013.

[8] X. Yu *et al.*, "Staring into the abyss: An evaluation of concurrency control with one thousand cores," *PVLDB*, vol. 8, no. 3, pp. 209–220, 2014.

[9] R. Agrawal *et al.*, "Concurrency control performance modeling: Alternatives and implications," *TODS*, vol. 12, no. 4, pp. 609–654, 1987.

[10] D. Tang *et al.*, "Adaptive concurrency control: Despite the looking glass, one concurrency control does not fit all," in *Proc. of CIDR*, 2017.

[11] T. Wang *et al.*, "Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores," *PVLDB*, vol. 10, no. 2, pp. 49–60, 2016.

[12] Z. Shang *et al.*, "Graph analytics through fine-grained parallelism," in *Proc. of SIGMOD*, 2016.

[13] A. Bestavros *et al.*, "Value-cognizant speculative concurrency control," in *Proc. of VLDB*, 1995.

[14] R. Johnson *et al.*, "Improving oltp scalability using speculative lock inheritance," *PVLDB*, vol. 2, no. 1, pp. 479–489, 2009.

[15] P.-A. Larson *et al.*, "High-performance concurrency control mechanisms for main-memory databases," *PVLDB*, vol. 5, no. 4, pp. 298–309, 2011.