Improving transaction abort rates without compromising throughput through judicious scheduling

Ana Nunes
High-Assurance Software Lab
INESC TEC & University of Minho
ananunes@di.uminho.pt

José Pereira
High-Assurance Software Lab
INESC TEC & University of Minho
jop@di.uminho.pt

ABSTRACT

Althought optimistic concurrency control protocols have increasingly been used in distributed database management systems, they imply a trade-off between the number of transactions that can be executed concurrently, hence, the peak throughput, and transactions aborted due to conflicts.

We propose a novel optimistic concurrency control mechanism that controls transaction abort rate by minimizing the time during which transactions are vulnerable to abort, without compromising throughput. Briefly, we throttle transaction execution with an adaptive mechanism based on the state of the transaction queues while allowing out-of-order execution based on expected transaction latency. Preliminary evaluation shows that this provides a substantial improvement in committed transaction throughput.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*concurrency, distributed databases*

General Terms

Performance, Reliability

Keywords

Optimistic concurrency control, adaptive scheduling

1. INTRODUCTION

Optimistic concurrency control is increasingly popular in distributed data management systems ranging from replicated relational databases [2, 3] to novel large scale and high throughput transactional systems such as Yahoo!'s OMID [5]. In these systems, potentially conflicting transactions are allowed to execute mostly without coordination. However, after execution, a certification (conflict detection) phase takes place to determine whether the resulting changes are consistent and should be applied to the database. While this mechanism allows more concurrency, transactions that are later found to conflict are aborted and must be re-executed.

Notice that the more transactions are allowed to execute concurrently, the more likely it is for conflicts to arise. Also, any transaction is vulnerable to being aborted by other transactions from the moment it starts to execute until it is certified, hence, the longer it takes to execute a given transaction, the more vulnerable it is. This is the main caveat of most optimistic concurrency control strategies: when loaded, latency increases and more conflicts are found. This can make it hard for long-running transactions to commit at all [1].

Current proposals fail to address this issue, as they typically either execute transactions as soon as these are submitted [2, 3, 5] or simply restrict the number of transactions that can be concurrently executed in the system [1]. We improve over them by applying a novel scheduling strategy that attempts to minimize aborts, while maximizing transaction throughput.

The main contribution of this paper is a transaction scheduling algorithm that uses the level of queuing in a system and transaction classification to self-adapt for optimal performance (high throughput, low abort rate). This is achieved by minimizing the time during which these are vulnerable to being aborted by concurrent transactions, thereby reducing the overall abort rate.

2. APPROACH

Our approach, AJITTS (Adaptive Just-In-Time Transaction Scheduling), is based on reaching and maintaining the optimal level of queueing in the system: as low as possible to minimize aborts but as high as needed to ensure that all resources are fully used to maximize throughput. This considers both that *(i)* transaction execution latency varies with load and that *(ii)* transactions that are expected to execute relatively faster should be scheduled proportionally later. The goal is that all transactions wait an equal and minimal amount of time after being executed, thus decreasing the likelihood of being aborted.

Briefly, transactions submitted to the system are totally ordered and placed in the queue. This applies both to a centralized queue [5] as well as to a consistenly replicated queue [2, 3, 1], that, for simplicity of presentation, we consider a single logical queue. At the head of the queue is the transaction currently undergoing certification, in the $certification$ state. All other transactions are in one of three states: $not\_executed$, $executing$, $executed$ or $aborted$.

Assuming first that all transactions take the same amount

SAC'13 March 18–22, 2013, Coimbra, Portugal

Table 1: Throughput and abort rate improvement

	Throughput	Abort Rate
OPT	51.4 tps	23.5%
AJITTS	150 tps	5.1%

of time to be executed, consider that there is a line in the queue that determines where transactions should start execution: all transactions before the line are not eligible to start executing, while all transactions between the line and the head of the queue that are in the *not_executed* state are to be executed. Simply put, transactions are evaluated for execution whenever transactions arrive to or leave the queue. Ideally, the line would be placed in such a position that each transaction's execution completes just as it arrives at the head of the queue, minimizing the time spent in the *executed* state before reaching *certification*, thus minimizing its vulnerability to being aborted by others. When a transaction reaches the head of the queue, if in the *executed* or *aborted* states, the transaction goes into *certification*. Conflict detection ensues: If the transaction was in the *executed* state, is immediately certified and, again, any conflicting transactions either in the *executing* or *executed* states are aborted.

AJITTS works as follows. If a transaction arrives to the head of the queue in the *aborted* state, it must be re-executed. In this case, the transaction has waited too long and the line should be pushed forward. On the other hand, if the transaction reaches the head of the queue in either *not_executed* or *executing* states, it cannot be certified until it finishes. Because certification must be in order, no other transaction can overtake it, leaving certification idle. In this case, the line should be pushed back. Moreover, AJITTS considers transactions with relatively different execution times by proportionally scaling the position of the line according to the expected duration of the transaction.

3. EVALUATION

AJITTS was evaluated using an event-driven simulator with execution traces obtained from a TPC-E-like benchmark on MySQL database. TPC-E is a benchmark that simulates the activities of a brokerage firm, handling customer account management, trade order execution on behalf of customers and the interaction with financial markets [4]. The trace was extracted from MySQL's binlog and provides the following information to the simulation: the timestamps at which each transaction started, how long it took to execute, and their write sets. It was observed that average transaction latency varies between 43 and 501 milliseconds depending on the type of transaction.

This simulator allows us to compare AJITTS with OPT, a protocol with a standard optimistic scheduler but with a conservative re-execution mechanism for previously aborted transactions [1]. Simply put, OPT optimistically schedules each execution as soon as it is submitted. Table 1 shows how the throughput achieved with AJITTS is much higher than with OPT for the same number of clients.

Figure 1 shows how the line positions evolve after each update. A transaction type's line position is updated whenever the estimate for its execution duration is changed or whenever the adaptation input parameter changes. For example, trade update transactions are scheduled much earlier when compared to other transaction types. Notice that

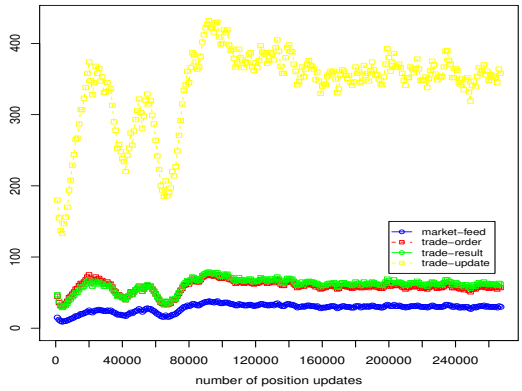

Figure 1: Line position per transaction type

the average duration for trade update transactions is significantly larger that the duration of the other transaction types. The position of the line for each transaction type converges quickly: for market feed, trade order and trade result, the line positions stabilize after 50000 committed transactions; for trade update transactions, the amplitude of the variation stabilizes after 80000 committed transactions.

4. CONCLUSION

When using optimistic concurrency control in replication protocols with conservative re-execution, each abort takes a toll on overall performance. We tackled this issue by proposing AJITTS, which combines an adaptive mechanism based on the state of transaction queues with out-of-order execution based on estimates of transaction latency.

AJITTS was evaluated using a TPC-E like benchmark having clearly outperformed a traditional replication protocol with optimistic concurrency control in both performance and latency figures. AJITTS performed better than the traditional protocol in terms of throughput and abort rate for workloads derived from the TPC-E like benchmark, but slightly different in terms of CPU availability and conflict probability.

5. REFERENCES

[1] A. Correia, J. Pereira, and R. Oliveira. Akara: A flexible clustering protocol for demanding transactional workloads. *On the Move to Meaningful Internet Systems: OTM 2008*, pages 691–708, 2008.

[2] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 134–143, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[3] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 14:71–98, 2003. 10.1023/A:1022887812188.

[4] Transaction Processing Performance Council (TPC). *TPC Benchmark E - Standard Specification*, revision 1.12.0 edition, June 2010.

[5] M. Yabandeh and D. Gómez Ferro. A critique of snapshot isolation. In *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, pages 155–168, New York, NY, USA, 2012. ACM.