TECHNICAL PERSPECTIVE: Checking Invariant Confluence, In Whole or In Parts

Johannes Gehrke
Microsoft Research
Redmond, WA; USA
johannes@acm.org

Never make a promise - you may have to keep it. — Neil Jordan

Database systems were known to provide strong consistency guarantees. As an example, database textbook defines the ACID guarantees as "four important properties of transactions to maintain data in the face of concurrent access and system failures" [2]. Beyond atomicity, consistency, and durability, the "I" in ACID is loosely defined as "Users should be able to understand a transaction without considering the effects of other concurrently executing transactions, even if the DBMS interleaves the actions of several transactions for performance reasons" [2]. In the resulting model called serializable execution, each transaction operates like it has the database to itself, and the result of running a set of transactions is equivalent to some serial execution of these transactions.

Serializable execution implies an ordering of transactions based on conflicts, where a conflict means that we have two transactions that are accessing the same database record, and at least one of them is a write. To ensure a serial ordering of the transactions, the conflicts must be totally ordered across transactions. In other words, if two transactions conflict, they need to coordinate. However, coordination has a price. If the database system is located within a single data center, coordination across nodes costs a few hundred microseconds, but if the system is wide-area distributed across several data centers, the delay between nodes may be in the 10s of milliseconds, restricting throughput to 10s of operations in the worst case. And in case the network is partitioned, the system may not be available at all.

For a while, most distributed systems dealt with this tradeoff in one of two ways. One popular option was to focus on high availability and low latency and perform the coordination asynchronously. Unfortunately this approach only provides weak consistency guarantees, so applications must use additional mechanisms such as compensation transactions or custom conflict resolution strategies, or they must restrict the programming model to eliminate the possibility of conflicts, for example by only allowing updates of a single record. Another approach was to insist on strong consistency and to accept slower response times because of coordination between nodes. It seemed like consistent semantics was not such a good idea after all.

In a recent landmark paper, Bailis et al asked the intriguing question: Are there cases where we may not need coordination between transactions at all, and thus we can achieve both high availability and low latency while maintaining application-level constraints [1]? The somewhat surprising answer is that in many practical scenarios the answer is yes. The paper introduced *invariant confluence*, a criterion that determines whether a set of transactions requires coordination for correct execution while maintaining integrity constraints. The framework requires developers to state the integrity constraints of the application on the database state a priori, but then it provides a necessary and sufficient condition for coordination-free execution. The resulting system only needs to coordinate in cases where the framework indicates that coordination is necessary; if it is possible, the framework guarantees that transactions do not violate any of the stated integrity constraints even if transactions do not coordinate. Analyzing TPC-C with invariant confluence by manually extracting the inherent constraints enabled Bailis et al. to show that only two of the twelve TPC-C constraints are not invariant confluent, and when applying the resulting insights to scaling TPC-C, they outperformed the previous best result of scaling TPC-C New-Order performance by factor of 25! However, coming up with the constraints and analyzing them is challenging as the authors admit themselves in the paper: "We have found the process of invariant specification to be non-trivial but feasible in practice;" [1].

The following paper is automating this manual process. The task of determining invariant confluence for an object given a set of transactions is no longer an exercise for the reader; instead, the paper provides an automatic way of checking for invariant confluence — a leap forward towards making the concept practical. The paper also goes a step further by taking objects that are not invariant confluent and in some cases allowing at least only occasional coordination instead of requiring coordination all the time. A beautiful set of results that is an important step towards scaling distributed database systems — with consistent semantics after all.

1. REFERENCES

[1] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. Coordination avoidance in database systems. *Proc. VLDB Endow.*, 8(3):185–196, 2014.

[2] R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.