Coordination Avoidance in Database Systems

Peter Bailis, Alan Fekete[†], Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica
UC Berkeley and [†]University of Sydney

ABSTRACT

Minimizing coordination, or blocking communication between concurrently executing operations, is key to maximizing scalability, availability, and high performance in database systems. However, uninhibited coordination-free execution can compromise application correctness, or consistency. When is coordination necessary for correctness? The classic use of serializable transactions is sufficient to maintain correctness but is not necessary for all applications, sacrificing potential scalability. In this paper, we develop a formal framework, invariant confluence, that determines whether an application requires coordination for correct execution. By operating on application-level invariants over database states (e.g., integrity constraints), invariant confluence analysis provides a necessary and sufficient condition for safe, coordination-free execution. When programmers specify their application invariants, this analysis allows databases to coordinate only when anomalies that might violate invariants are possible. We analyze the invariant confluence of common invariants and operations from real-world database systems (i.e., integrity constraints) and applications and show that many are invariant confluent and therefore achievable without coordination. We apply these results to a proof-of-concept coordination-avoiding database prototype and demonstrate sizable performance gains compared to serializable execution, notably a 25-fold improvement over prior TPC-C New-Order performance on a 200 server cluster.

1. INTRODUCTION

Minimizing coordination is key in high-performance, scalable database design. Coordination—informally, the requirement that concurrently executing operations synchronously communicate or otherwise stall in order to complete—is expensive: it limits concurrency between operations and undermines the effectiveness of scale-out across servers. In the presence of partial system failures, coordinating operations may be forced to stall indefinitely, and, in the failure-free case, communication delays can increase latency [9, 28]. In contrast, coordination-free operations allow aggressive scale-out, availability [28], and low latency execution [1]. If operations are coordination-free, then adding more capacity (e.g., servers, processors) will result in additional throughput; operations can execute on the new resources without affecting the old set of resources. Partial failures will not affect non-failed operations, and latency between any database replicas can be hidden from end-users.

Unfortunately, coordination-free execution is not always safe. Uninhibited coordination-free execution can compromise application-level correctness, or consistency.[1] In canonical banking application examples, concurrent, coordination-free withdrawal operations can result in undesirable and "inconsistent" outcomes like negative account balances—application-level anomalies that the database should prevent. To ensure correct behavior, a database system must coordinate the execution of these operations that, if otherwise executed concurrently, could result in inconsistent application state.

This tension between coordination and correctness is evidenced by the range of database concurrency control policies. In traditional database systems, serializable isolation provides concurrent operations (transactions) with the illusion of executing in some serial order [15]. As long as individual transactions maintain correct application state, serializability guarantees correctness [30]. However, each pair of concurrent operations (at least one of which is a write) can potentially compromise serializability and therefore will require coordination to execute [9, 21]. By isolating users at the level of reads and writes, serializability can be overly conservative and may in turn coordinate more than is strictly necessary for consistency [29, 39, 53, 58]. For example, hundreds of users can safely and simultaneously retweet Barack Obama on Twitter without observing a serial ordering of updates to the retweet counter. In contrast, a range of widely-deployed weaker models require less coordination to execute but surface read and write behavior that may in turn compromise consistency [2, 9, 22, 48]. With these alternative models, it is up to users to decide when weakened guarantees are acceptable for their applications [6], leading to confusion regarding (and substantial interest in) the relationship between consistency, scalability, and availability [1, 9, 12, 18, 21, 22, 28, 40].

In this paper, we address the central question inherent in this tradeoff: when is coordination strictly necessary to maintain application-level consistency? To do so, we enlist the aid of application programmers to specify their correctness criteria in the form of *invariants*. For example, our banking application writer would specify that account balances should be positive (e.g., by schema annotations), similar to constraints in modern databases today. Using these invariants, we formalize a *necessary* and sufficient condition for invariant-preserving and coordination-free execution of an application's operations—the first such condition we have encountered. This property—invariant confluence ($\mathcal{I}$-confluence)—captures the potential scalability and availability of an application, independent of any particular database implementation: if an application's operations are $\mathcal{I}$-confluent, a database can correctly execute them without coordination. If operations are not $\mathcal{I}$-confluent, coordination is required to guarantee correctness. This provides a basis for *coordination avoidance*: the use of coordination only when necessary.

While coordination-free execution is powerful, are any *useful* operations safely executable without coordination? $\mathcal{I}$-confluence analysis determines when concurrent execution of specific operations can be "merged" into valid database state; we accordingly

 Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, Aug. 31st - Sept. 4th, 2015, Kohala Coast, Hawaii.
Proceedings of the VLDB Endowment, Vol. 8, No. 3

[1] Our use of the term "consistency" in this paper refers to *application-level* correctness, as is traditional in the database literature [15, 21, 25, 30, 56]. As we discuss in Section 5, replicated data consistency (and isolation [2, 9]) models like linearizability [28] can be cast as application criteria if desired.