Conditional Conflict Serializability—An Application Oriented Correctness Criterion

Ole J. Anfindsen
Telenor R&D, Norway

Serializability is too strict a correctness criterion for several application domains, in particular where support for long-lasting transactions is required. This paper describes a generalized version of serializability called conditional conflict serializability (CCSR), which is built on a customized notion of conflict rather than the standard commutativity-based one. The actual customization of conflicts is carried out by applications that associate parameters with their read and write operations. The semantics of such parameters are user-defined, and can be chosen to suit various needs. CCSR can be enforced by means of two phase locking with parameterized locks. Transaction histories that are strict or rigorous modulo CCSR are defined, showing that a CCSR scheduler need not rely on compensating actions for recovery.

Standard transaction management ensures the ACID properties for transactions (atomicity, consistency, isolation, and durability; Hörder & Reuter 1983, Gray & Reuter 1993). Of primary interest here is the isolation property, which corresponds to the correctness criterion serializability. It is generally acknowledged in the database research community that serializability is unsuitable for several application domains (Elmagarmid 1992, Kaiser 1995, Ramamritham & Chrysanthi 1996), in particular where transaction duration is long relative to response time requirements. This paper presents conditional conflict serializability (CCSR), a correctness criterion capable of supporting serializability, dirty reads, and almost any degree of concurrency in between those two extremes. This makes CCSR suitable for almost any application domain where read-write and write-read conflicts need to be handled in a more lenient way than with unconditional conflict serializability. In particular, CCSR is appropriate for applications that need long-lasting transactions.

An important motivation behind CCSR is the need for a correctness criterion that is both general and simple. There is need for a framework that offers application programmers easy-to-use customization of concurrency control, without affecting programmers who need their applications to be fully isolated from others. Another major motivation behind CCSR has been to enable arbitrary levels of concurrency without forcing programmers to perform recovery by means of compensating actions.

The larger context for the results presented in this paper is Apotram (which stands for application-oriented transaction model; Anfindsen 1997). In particular, it should be noted that while CCSR has nothing to offer in terms of handling write-write conflicts, it is part of a framework where this is taken care of. The Apotram mechanism for handling write-write conflicts is called nested databases (Anfindsen 1996; 1997, 45-52), and the corresponding correctness criterion is called nested conflict serializability (NCSR). NCSR and CCSR can be combined into a single correctness criterion.

Related Work

The ideas set forth in this paper are related to those of Farrag & Özsu (1989) and their correctness criterion Relatively Consistent Schedules (RELC), which is a
generalization of the correctness criteria of Garcia-Molina (1983) and Lynch (1983). All these references, just like this paper, are aimed at allowing greater concurrency than is possible with serializability. A detailed analysis of the differences and similarities of RELC and CCSR can be found in (Anfindsen 1997, 93-96), the main conclusion of which is that a CCSR scheduler can produce all RELC histories except those that are inherently unrecoverable. This is a reflection of the different approaches taken to recovery; while (Garcia-Molina; 1983; Farrag & Özsu; 1989; Lynch, 1983) all rely on compensating actions, the Apotram/CCSR approach is based on the premise that only as a last resort should one be willing to compromise the commitment atomicity of transactions, because we then also sacrifice their recoverability. In other words, this paper, like the just mentioned references, aims at increased concurrency, but without giving up the benefits of system-controlled recovery.

Ammann et al (1997) aim at increased concurrency by means of semantic-based decomposition of transactions. The focus of their paper is on formal methods used to obtain correct decompositions. The model of Ammann et al is closely related to, but more general than, the one from Farrag & Özsu (1989). Both models rely on recovery by means of compensating actions.

Shasha et al (1995) investigate how to increase concurrency in database systems by chopping up transactions based on information supplied by a database system user. Their goal is to ensure serializability “without paying for it.” According to Ammann et al (1997, 240), the approach of Shasha et al is less general than their own, “Since they do not use any semantic knowledge.”

Kirsch et al (1994a) introduce Database Conversations, which is “an application-independent, tight framework for jointly modifying common data.” In their model each data unit has a binary conversation flag associated with it, and the two make up an indivisible unit; “the information whether a data unit is uncertain or not, is stored explicitly with the data item itself and not implicitly in some transaction semantics.” If the conversation flag is set, it means that a conversation context has been created for the data item in question. Conversation contexts are used for coordinating updates to the data item from multiple transactions, allowing one to start from the old value, step over a number of intermediate values, and eventually arrive at the final value. During this, the data item can be read by other transactions, which will then see its old value but will also be told by the conversation flag that this value is unreliable.

The conversation flags of (Kirsch et al., 1994a) correspond somewhat to Apotram’s notion of access mode parameters. Both are used to tell other transactions something about the status of data, both are examples of the use of metadata, and both are based on the notion of uncommitted data being uncertain. But while conversation flags have only two values, access mode parameters can have an arbitrary number. Thus, this paper, like (ibid), is built on the observation that uncommitted data is unreliable, and that transactions therefore should communicate with each other on the status of such data. The uncertainty associated with uncommitted data is further elaborated in (Kirsch et al., 1994b). While the language for intertransaction communication in (Kirsch et al., 1994a) is based on dynamically created conversation contexts, CCSR is based on attaching parameters to read and write operations.

Muth (1997) “propose a transaction model for multidatabase systems - the heterogeneous 3-level-transaction model - which can be parameterized to support different compromises between global serializability and local autonomy, and fully supports recovery of global transactions.” (ibid, 358, emphasis in original). This model is based on multi-level transactions (Weikum, 1991). Just as Apotram/CCSR, the heterogeneous 3-level-transaction model regards the commutativity-based notion of conflict as being too restrictive in many cases. Four different kinds of information is exploited in order to provide “different alternatives for conflict definitions between local transactions and global subtransactions” (Muth 1997, 359). These are system architecture, semantics of actions, actions allowed, and consistency constraints (ibid, 371-372). The importance of this is commented as follows: “This is the key of our work, as it allows tailoring the model to specific application needs.” (ibid, 359). The heterogeneous 3-level-transaction model is clearly more complicated than Apotram/CCSR, but Muth remarks that “most of the complexity is in the theoretical foundation” and that “from an application point of view” using his model will be “rather simple” (ibid, 396). Another difference from Apotram/CCSR is that Muth relies on recovery by means of “inverse actions,” i.e., compensation.

Agrawal et al (1993) introduce three new correctness criteria; consistency, orderability, and strong orderability. Consistency is based solely on database consistency assertions specified by the users, and the users must make sure assertions are chosen in such a way that “nothing unexpected happens” (ibid, 469). Orderability and strong orderability are, just like CCSR, based on notions of equivalence with serial histories, and are generalizations of view and conflict serializability, respectively. Agrawal et al sum up the main differences between their own model and traditional models, of which Apotram/CCSR is an example, as follows (ibid, 482-483): “First, our model assumes abstract data types, which support semantically rich operations, whereas the traditional model is limited to read and write operations. Second, in our model users specify predicates on the database through consistency assertions before every operation.”

Korth & Speegle (1994) introduce a transaction model called nested transactions with predicates and versions, or NT/PV, which generalizes conflict serializability. In this model each update operation generates a new version of the data item being updated, and the model relies on the use of explicit predicates to form pre- and postconditions for transactions.