Fundamentals of Transactional Information Systems: Theory, Algorithms, and Practice of Concurrency Control and Recovery

by

Gerhard Weikum, MPI Saarbrücken, Germany
Gottfried Vossen, University of Münster, Germany

List of Errata

June 27, 2004

The following list of errate has been compiled from numerous sources, including colleagues and students who have provided us with a number of hints to items and points that are unclear, typos, and other stuff. We appreciate all this input and to try make use of it in the next edition of the book. In the meantime, this list should help our readers in case questions arise.

1 Errata

Chapter 3

- page 109, Figure 3.13: The schedules s_3 and s_5 need to be swapped, i.e., it is $s_3 \notin$ VSR and $s_5 \in$ VSR.
  In proof, note that $s_3 \in$ FSR since it is equivalent to $t_2 t_1$. However, $\mathrm{RF}(s_3) = \{(t_1, x, t_2), (t_1, x, t_\infty), (t_1, y, t_\infty)\}$. On the other hand, $\mathrm{RF}(t_2, t_1)$ contains tuple (t_0, x, t_2), while $\mathrm{RF}(t_1 t_2)$ contains (t_2, y, t_∞),

so each shows at least one difference from RF(s_3). Thus, $s_3 \notin$ VSR. Note further that $s_3 \in$ CMFSR.

Chapter 5

- page 199: Change Theorem 5.6 as follows: If a multiversion history $m = (\text{op}, <)$ is in MCSR, then there exists a version function f' such that the multiversion history $m' = (\text{op}, <)$ that has the same operations and operation ordering as m but may use a different version function than f' is in MVSR.
- page 208: In the second line of the schedule of Example 5.10 insert $w_2(x)$ between $wl_2(x)$ and $cl_2(x)$ and replace $wl_3(y)$ by $wl_3(z)$.
- page 211: The first two sentences of Item 1. near the bottom of the page need to be changed into: *Update transactions are subject to the strong SS2PL protocol. They acquire conventional locks for both read and write steps, which are released according to the strong two-phase rule, with both read and write locks held until commit.*

Chapter 7

- pp. 255–259, Figures 7.3 through 7.6: Remove the L_0 operation $w(p)$ from the L_1 operations Fetch(x), Fetch(y), and Fetch(z).
  Also, add "and" in the Where clauses of the Select statements where two conditions occur (e.g., *Select Name from Persons where City = "Seattle" and Age = 29* in Figure 7.4).

Chapter 9

- page 346, Exercise 9.6: In the right subtree of the root, the arc from entry 24 in node p_2 should go down to node q_5, and the arc from 24 to q_4 as well as the one from q_4 to q_5 should be eliminated.

Chapter 11

- page 392, Example 11.11: Replace "we can verify that $s_2 \in$ ACA" by "we can verify that $s_3 \in$ ACA."

Chapter 12

- page 440, lines 7 and 6 from the bottom (Item 4 of itemized list): Replace "Therefore, it is also known as the class of no-steal no-force algorithms." by "*Therefore, it is also known as the class of steal no-force algorithms.*"

Chapter 13

- page 524: In the redo pass algorithm, remove line 5 from the bottom (the page fetch is unnecessary before testing the DirtyPages table).
- page 528: In Figure 13.21 the line "21: compensate(18, write(r, t_3))" must be replaced by "21: compensate(19, write(r, t_3))."

Chapter 15

- page 567: In the 3rd paragraph, insert after the sentence "... there exists a redo dependency ... if f precedes g on the log and there exists a page x such that $x \in$ readset(f) and $x \in$ writeset(g)." the following sentence:
  We assume that the order of the two logical log entries reflects the serialization order between the corresponding subtransactions.

Chapter 16

- page 598, last paragraph (starting with "The archive log simply collects ..."): Replace the entire paragraph by the following:

  The archive log simply collects all regular log entries of the stable log (maintained for crash recovery) since the last backup. This includes page-level redo/undo log entries as well as high-level undo log entries for object-model transactions, and also compensation log entries (CLEs) for aborted and undone transactions. The archive log is conceptually separated from the stable log: the archive log serves to protect the system against media failures, taking crash recovery for granted, and the stable log merely serves the purpose of crash recovery. As the archive log covers an extended time period up to the present, its log entries form a superset of the stable log. The implementation could simply replicate *the stable log on independent disk(s) or tape(s9) and periodically copy the recent tail of the stable log to the archive log. Alternatively, all log entries that are forced from the log buffer could be simultaneously written to the stable log and the archive log. For the first, copy-based, approach with a replicated stable log, it is convenient to organize the stable log into a set of files that are used in a round-robin fashion so that currently passive files can be copied without interfering with the I/Os on the stable log. The begin and end of when a backup is taken is recorded in the log by creating special "begin-backup" and "end-backup" log entries.*

- page 599, first paragraph: Replace "stable log" by "archive log" in lines 2 and 4.

Chapter 19

- page 742: In the paragraph following Lemma 19.1, insert after the sentence "The protocol that ensures independent recovery is a variation of 2PC ... in between the coordinator's Collecting and Committed states." the following sentences, and start the next sentence as a new paragraph:
  For simplicity, the following considers only a two-party protocol with one participant and the coordinator. The arguments can be generalized, but this extension is nontrivial.

- page 765, Exercise 19.1: Add the following sentence: *For simplicity, you may restrict yourself to the two-party case, with one participant and the coordinator.* The generalization to multiple participants requires nontrivial extensions.

2 Comments

Parallel transactions, in particular parallel changes by multiple threads of a single transaction, require lock compatibility among threads. Using a separate sub-transaction per thread is a mess, because separate threads may read & write a B-tree entry, for example. The simple solution is to use latches to separate threads (and their accesses to in-memory data structures, including images of disk pages in the buffer) and to use lock to separate transactions (and their accesses to logical database contents). Concurrency control on catalogs, and how all that interacts with recompilation, plan caching & invalidation, etc. Maybe it's as simple as using read & write locks on catalog records, but it seems that (at least our) implementations make this much more complex, presumably because nobody in the "academic" community has thought this through and described a simple, clear, coherent, and comprehensive model. For example, SQL Server has special lock modes for tables, e.g., "schema stability," "schema modify," "bulk insert," "truncate," etc. A mess. Releasing locks within a transaction seems utterly suspect, e.g., a range lock once a newly inserted key is locked. Maybe a useful model uses a system transaction to insert an invalid record with the desired key (and commits & releases locks), and then a user transaction makes this entry valid etc. This is very much parallel to deletion with ghost records that won't really get deleted until after the user transaction has committed.

3 Typos

- page xxiv, last two lines: Replace "vossen@uni-mvenster.de" by "vossen@uni-muenster.de"
- page 428, line 11 from bottom: Replace "tandem" by "Tandem"
- page 692, line before Example 18.7: Replace "seen" by "seem"
- page 807, bibliographic entry for Jablonski and Bussler: Change the year from "1999" to "1996"

4 Updates

- page xxiv, last two lines: Replace "weikum@cs.uni-sb.de" by "weikum@mpi-sb.mpg.de"
- page 813: Mehrotra, S., R. Rastogi, Y. Breitbart, H.F. Korth, A. Silberschatz (2000): Overcoming Heterogeneity and Autonomy in Multidatabase Systems. *Information and Computation* **167**, pp. 137–172.