The Serial Safety Net: Efficient Concurrency Control on Modern Hardware

Seminarausarbeitung

Florian Lüdiger
Technische Universität Dortmund
florian.luediger@tu-dortmund.de

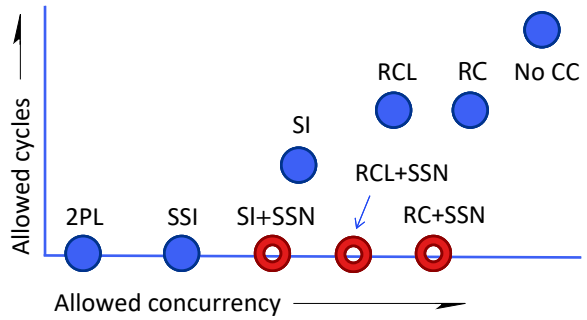

Abbildung 1: Klassische Concurrency-Control-Verfahren im Vergleich zum Serial Safety Net

1 EINLEITUNG

In diesem Dokument wird das in der Veröffentlichung „The Serial Safety Net: Efficient Concurrency Control on Modern Hardware“ von Wang et al. [5] vorgestellte Verfahren zur Sicherstellung der Serialisierbarkeit von Transaktionsplänen erläutert. Dabei werden bestehende Verfahren für Concurrency-Control(CC) wie beispielsweise Snapshot-Isolation(SI), Read-Committed(RC) oder Read-Committed mit Locks(RCL) so erweitert, dass diese die Serialisierbarkeit der entstehenden Pläne gewährleisten.

Der Vorteil des Serial Safety Nets(SSN) gegenüber klassischen Verfahren, wie dem Zwei-Phasen-Sperrprotokoll(2PL) oder der Serializable-Snapshot-Isolation(SSI), besteht darin, dass eine bessere Nebenläufigkeit von Transaktionen ermöglicht wird, wodurch der Durchsatz des gesamten Systems massiv gesteigert wird.

In Abbildung 1 wird schematisch dargestellt, dass ein bestimmter Trade-off zwischen der zugelassenen Nebenläufigkeit und den erlaubten Zyklen im Abhängigkeitsgraphen, also dem gewünschten Isolationslevel, besteht. Klar erkennbar ist, dass ein optimales Concurrency-Control-Verfahren keinerlei Zyklen erlaubt und dennoch eine maximale Nebenläufigkeit gewährleistet. Es wird deutlich, dass die durch das Serial Safety Net erweiterten Verfahren überhaupt keine solcher Zyklen erlauben und somit in dieser Hinsicht gleichwertig zu Verfahren wie 2PL und SSI sind. Gleichzeitig ist allerdings erkennbar, dass die erlaubte Nebenläufigkeit wesentlich höher ist und somit ein Performanzgewinn erwartet wird.

Seminar Transaktionsverwaltung auf moderner Hardware, Wintersemester 2017/18, Technische Universität Dortmund
2018.

Einige Gründe dafür, dass die bisher bekannten Verfahren zur Sicherstellung der Serialisierbarkeit, wenig performant sind, finden sich darin, dass diese einen hohen Overhead produzieren und teilweise zu unnötigen Transaktionsabbrüchen führen, wodurch möglicherweise gültige Pläne verworfen werden. Außerdem funktionieren diese aufgrund schlechter Skalierung schlecht auf moderner Hardware, bei der immer mehr Operationen im Hauptspeicher stattfinden, wodurch die I/O-Last sinkt und damit ein noch größerer Fokus auf einem effizienten Concurrency-Control-System liegt.

Wichtig ist in diesem Zusammenhang zu erwähnen, dass ein Transaktionsplan, der in diesem Dokument als serialisierbar bezeichnet wird, keinen Schutz gegen Phantome bietet. Das Serial Safety Net lässt sich durch das Verwenden von Sperren allerdings leicht erweitern, sodass das Vorkommen von Phantomen ausgeschlossen wird, worauf in einem die ursprüngliche Veröffentlichung ergänzenden Artikel näher eingegangen wird. [6] Weitere Informationen zu möglichen Anomalien, Isolationsleveln und dem Phantom-Problem finden sich in [3].

2 THE SERIAL SAFETY NET

Das Serial Safety Net baut auf bestehenden Multiversion-Concurrency-Control-Verfahren auf. In einem System, welches Multiversion-Concurrency-Control(MVCC) verwendet, bestehen alle Datenbankelemente aus einer Sequenz von Versionen, wobei Schreiboperationen jeweils eine Version anlegen und Leseoperationen eine Version zurückgeben.

Für solche MVCC-Verfahren sichert das Serial Safety Net einen kreisfreien Abhängigkeitsgraphen, wodurch die Serialisierbarkeit des Transaktionsplans gewährleistet wird. Der Abhängigkeitsgraph stellt dabei eine Übersicht über die Abhängigkeit zwischen den Transaktionen eines Plans dar, wobei die Knoten des Graphen die committeten Transaktionen und die Kanten die Abhängigkeiten zwischen diesen darstellen. Ein Graph ohne Zyklen garantiert dabei immer, dass ein äquivalenter serieller Plan zu den ausgeführten Transaktionen existiert, welcher zum selben Ergebnis geführt hätte.

Definition 1. *Die Abhängigkeit $T \leftarrow U$ zwischen den Transaktionen T und U besagt, dass U von T abhängig ist und somit T als direkter Vorgänger und U als direkter Nachfolger bezeichnet wird.*

Es gibt zwei verschiedene Arten von Abhängigkeiten zwischen Transaktionen:

- $T_i \xleftarrow{w:x} T$ - **Lese-/Schreibabhängigkeit**: T greift auf eine Version zu, welche T_i erstellt hat, weshalb T nach T_i serialisiert werden muss

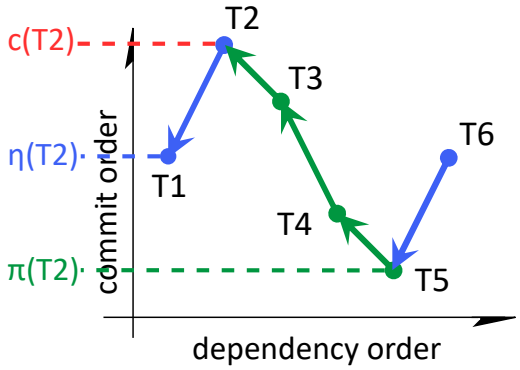

Abbildung 2: Veranschaulichung von Back- und Forward-Edges sowie den Zeitstempeln von Transaktion $T2$

- $T \xleftarrow{r:w} T_j$ - **Anti-Abhängigkeit**: T liest eine Version, die T_j überschrieben hat, weshalb T vor T_j serialisiert werden muss

Eine zentrale Rolle bei der Umsetzung des Serial Safety Nets spielt die Untersuchung der Abhängigkeiten von Transaktionen in Verbindung mit der Commit-Reihenfolge. Dazu lassen sich zwei verschiedene Typen von Abhängigkeiten wie folgt definieren.

Definition 2. *Bei einer Back-Edge $T \xleftarrow{b} U$ committet der Nachfolger U zuerst.*

Definition 3. *Bei einer Forward-Edge $T \xleftarrow{f} U$ committet der Vorgänger T zuerst.*

Definition 4. *Eine reflexive, transitive Back-Edge $T \xleftarrow{b*} U$ bezeichnet eine Verbindung, bei der T von U ausschließlich über Back-Edges erreichbar ist.*

Zum besseren Verständnis sind die genannten Begriffe in Abbildung 2 veranschaulicht worden. Die abgebildeten Back-Edges ergeben zusammen eine reflexive, transitive Back-Edge von Transaktion $T2$ zu $T5$.

Die Umsetzung des Serial Safety Nets erfordert das Erfassen der folgenden drei Zeitstempel zu jeder Transaktion, welche es später erlauben einen Abhängigkeitszyklus zu erkennen. Die vorgestellten Zeitstempel sind ebenfalls in Abbildung 2 für die Transaktion $T2$ eingezeichnet.

Definition 5. *$c(T)$ bezeichnet den Commit-Zeitpunkt der Transaktion T*

Definition 6. *$\pi(T)$ bezeichnet den Commit-Zeitpunkt des ältesten Nachfolgers, der durch Back-Edges erreichbar ist:*

$$\pi(T) = \min(c(U) : T \xleftarrow{b*} U) = \min(\{\pi(U) : T \xleftarrow{b} U\} \cup \{c(T)\})$$

Definition 7. *$\eta(T)$ bezeichnet den Commit-Zeitpunkt des zuletzt committeten Vorgängers von T:*

$$\eta(T) = \max(\{c(U) : U \xleftarrow{f} T\} \cup \{-\infty\})$$

Mithilfe dieser Zeitstempel lässt sich nun für jede Transaktion T ein sogenanntes Ausschlussfenster definieren, welches garantiert, dass ein Vorgänger von T nicht gleichzeitig ein Nachfolger sein kann, was auf einen Abhängigkeitskreis hinweisen würde. Eine Verletzung dieses Ausschlussfensters durch eine Abhängigkeit $U \leftarrow T$ lässt sich feststellen, wenn ein Vorgänger U gefunden wird, sodass die folgende Ungleichung erfüllt ist.

$$\pi(T) \leq c(U) \leq c(T) \tag{1}$$

Gibt es für die Transaktion T also einen Vorgänger U, welcher nach dem ältesten Nachfolger von T, nämlich $\pi(T)$, committet wurde, dann kann nicht sichergestellt werden, dass U nicht auch ein Nachfolger von T ist. Dies liegt daran, dass die Transaktion nichts über die Nachfolger von $\pi(T)$ weiß und es somit möglich wäre, dass U einer dieser Nachfolger ist, was damit einen Abhängigkeitszyklus bedeuten würde.

In dem Beispiel aus Abbildung 2 ist erkennbar, dass für die Transaktion $T2$ eine solche Verletzung des Ausschlussfensters vorliegt, da die Transaktion $T1$, welche ein Vorgänger von $T2$ ist, zwischen $c(T2)$ und $\pi(T2)$ committet wurde. Dies wird außerdem klar dadurch, dass die Transaktion $T6$ in diesem Beispiel ebenfalls die Transaktion $T1$ sein könnte, wovon $T2$ nicht direkt etwas wüsste. Damit wäre der Abhängigkeitszyklus entstanden und der Plan nicht serialisierbar.

Um die Umsetzung des Serial Safety Nets einfacher und performanter zu gestalten, lässt sich die Ungleichung 1 noch weiter vereinfachen. Zum einen müssen nur Vorgänger betrachtet werden, welche vor T committet wurden, da ansonsten der zweite Teil der Ungleichung automatisch nicht erfüllt wäre. Dies eröffnet die Freiheit das Überprüfen des Ausschlussfensters erst zum Commit-Zeitpunkt der betrachteten Transaktion zu durchzuführen.

Außerdem wird nur der Vorgänger von T betrachtet, dessen Commit-Zeitpunkt am spätesten ist, da dieser maßgebend für das Erfüllen des ersten Teils der Ungleichung ist. Wie vorher beschrieben wird die Commit-Zeit des zuletzt committeten Vorgängers von T mit $\eta(T)$ bezeichnet, wodurch die Ungleichung 1 folgendermaßen vereinfacht wird.

$$\pi(T) \leq \eta(T) \tag{2}$$

Wird diese Bedingung auf das in Abbildung 2 beschriebene Beispiel angewendet, wird deutlich, dass die Ungleichung für Transaktion $T2$ erfüllt ist und somit eine Verletzung des Ausschlussfensters erkannt wird.

Eine weitere hervorragende Eigenschaft des Serial Safety Nets ist die Möglichkeit des sogenannten **Safe Retry**. Wenn eine Transaktion aufgrund einer Verletzung des Ausschlussfensters abgebrochen werden muss, so ist es durch diese Eigenschaft möglich dieselbe Transaktion direkt zu wiederholen, ohne dass derselbe Konflikt erneut auftreten kann. Als Beispiel sei dazu angenommen, dass die Transaktion T wegen einer Ausschlussfensterverletzung von Transaktion U abgebrochen werden muss. Bei erneutem Durchführen der Transaktion als T' kann dieselbe Verletzung durch U nicht auftreten, da der Commit-Zeitpunkt von U vor dem Anfang der Transaktion T' liegt und somit nach 1 keine Verletzung zu befürchten ist.

3 EVALUATIONSUMGEBUNG

Um die nachfolgende Evaluierung des vorgestellten Verfahrens verstehen zu können, ist ein genaueres Verständnis der Evaluationsumgebung erforderlich. Besonders interessant sind hierbei die verglichenen Concurrency-Control-Verfahren und der verwendete Benchmark.

3.1 Verwendete CC-Verfahren

Für die Bewertung des Serial Safety Nets wurde dieses auf die Concurrency-Control-Verfahren Read Committed und Snapshot Isolation angewendet, welche beide normalerweise keine Serialisierbarkeit gewährleisten. Verglichen wurden diese dann mit der ursprünglichen Implementierung der Snapshot Isolation und der Optimistic-Concurrency-Control, welche beide keine Serialisierbarkeit sichern. Außerdem wurde ein Vergleich mit der Serializable Snapshot Isolation durchgeführt, welche vorher dazu verwendet wurde die Serialisierbarkeit bei der Verwendung von Snapshot Isolation zu gewährleisten. Nachfolgend werden einige Hinweise zu den verwendeten Verfahren erläutert, welche für das Verständnis der Evaluation erforderlich sind.

Read Committed (RC): Dieses Verfahren ist in Form eines Isolationslevels sehr bekannt und weit verbreitet, es finden sich daher auch viele Informationen dazu wie beispielsweise in [3]. Es wird dabei immer die neueste, committete Version eines Datensatzes gelesen und daher niemals blockiert. Bei Schreibvorgängen wird eine neue Version angelegt, die die vorhergehende Version ersetzt, was nur blockiert, falls die vorherige Version noch nicht committet wurde. Durch dieses Verfahren werden Abhängigkeitszyklen erlaubt, allerdings werden dirty reads und lost writes [3] dadurch verhindert.

Snapshot Isolation (SI): Die Snapshot Isolation führt alle Leseoperationen einer Transaktion auf einem Snapshot vom Beginn der Transaktion aus, wodurch deren gesamte Leseoperationen denselben konsistenten Zustand sehen. Enthält eine Transaktion T Schreiboperationen, so darf diese nur committen, wenn es keine Transaktion U gibt, sodass

- U zwischen Start- und Endzeitpunkt von T committet, und
- U ein Datenobjekt manipuliert hat, welches T ebenfalls manipuliert hat.

Das Verfahren lässt sich nicht wie beispielsweise Read Committet in die Standard-ANSI-Isolationslevel einordnen [2], eine vollständige Serialisierbarkeit ist allerdings nicht gewährleistet, denn es kann der sogenannte Write Skew auftreten.

Dabei wird wie in Abbildung 3 dargestellt über Kreuz auf Werte zugegriffen, wodurch ein anderes Ergebnis eintreten kann als bei dem korrespondierenden seriellen Plan. Im Beispiel gibt es eine Transaktion, welche alle weißen Kugeln schwarz färbt und eine andere Transaktion, welche alle schwarzen Kugeln weiß färbt. In einem seriellen Plan würden zunächst alle weißen Kugeln schwarz gefärbt, wodurch sämtliche Kugeln schwarz gefärbt wären. Daraufhin würden alle Kugeln von der anderen Transaktion weiß gefärbt werden, was dazu führen würde, dass sämtliche Kugeln dieselbe Farbe hätten. Bei Verwendung der Snapshot Isolation würden die Transaktionen gegebenenfalls parallel laufen, was dazu führen würde, dass jede Transaktion den Anfangszustand der Kugeln sehen würden, woraufhin jede Transaktion die für sie relevante Hälfte der

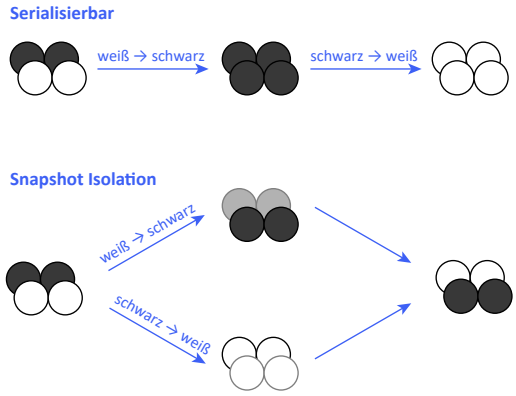

Abbildung 3: Beispiel für das Problem des Write Skew bei der Snapshot Isolation [4]

Kugeln umfärben würde. Nach dem Commit beider Transaktionen wäre nun immer noch die Hälfte aller Kugeln weiß und die andere Hälfte schwarz, die Kugeln hätten nur ihre Farbe getauscht, was bei einem seriellen Transaktionsplan nicht auftreten könnte.

Serializable Snapshot Isolation (SSI): Durch den geschickten Einsatz von Sperren, wird das Auftreten der vorhergehen vorgestellten Problematik bei der Snapshot Isolation verhindert. Somit sind bei der Verwendung dieses Verfahrens keinerlei Abhängigkeitszyklen möglich.

3.2 Der TPC-C Benchmark

Um die Geschwindigkeitsvorteile des Serial Safety Nets gegenüber den klassischen Verfahren quantifizieren zu können, wurde der TPC-C Benchmark verwendet.[1] Es handelt sich dabei um einen Online Transaction Processing (OLTP) Benchmark, welcher mehrere Transaktionstypen und eine komplexe Datenbank bietet. Die insgesamt fünf verschiedenen Transaktionsarten modellieren dabei die Alltagsaktivitäten eines Großhandels, was das Verwalten und Ausliefern von Bestellungen, das Überwachen von Zahlungen, das Abfragen des Bestellstatus und das Beobachten des Warenbestandes umfasst. Damit stellt dieser Benchmark eine hervorragende Simulation von stark verbreiteten Anwendungsgebieten dar, welche die Performanz des getesteten Systems in vielen Alltagssituationen einschätzen lässt.

4 EVALUATION

5 FAZIT

6 PERSÖNLICHE MEINUNG

LITERATUR

[1] [n. d.]. TPC-C is an On-Line Transaction Processing Benchmark. ([n. d.]).
[2] A. Adya, B. Liskov, and P. O'Neil. 2000. Generalized isolation level definitions. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. 67–78. https://doi.org/10.1109/ICDE.2000.839388
[3] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. 1995. *A Critique of ANSI SQL Isolation Levels*. Technical Report. https://www.microsoft.com/en-us/research/publication/a-critique-of-ansi-sql-isolation-levels/
[4] Craig Freedman. 2007. Serializable vs. Snapshot Isolation Level. (2007). https://blogs.msdn.microsoft.com/craigfr/2007/05/16/serializable-vs-snapshot-isolation-level/

[5] Tianzheng Wang, Ryan Johnson, Alan Fekete, and Ippokratis Pandis. 2015. The Serial Safety Net: Efficient Concurrency Control on Modern Hardware. In *Proceedings of the 11th International Workshop on Data Management on New Hardware (DaMoN'15)*. ACM, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/2771937.2771949

[6] Tianzheng Wang, Ryan Johnson, Alan Fekete, and Ippokratis Pandis. 2016. Efficiently making (almost) any concurrency control mechanism serializable. *CoRR* abs/1605.04292 (2016). arXiv:1605.04292 http://arxiv.org/abs/1605.04292