

The Serial Safety Net: Efficient Concurrency Control on Modern Hardware

Seminarausarbeitung

Florian Lüdiger

Technische Universität Dortmund

florian.luediger@tu-dortmund.de

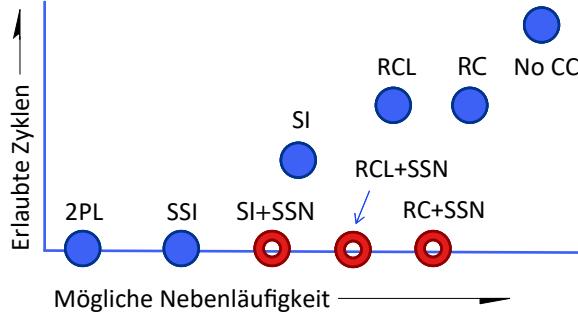


Abbildung 1: Klassische Concurrency-Control-Verfahren im Vergleich zum Serial Safety Net

1 EINLEITUNG

In diesem Dokument wird das in der Veröffentlichung „The Serial Safety Net: Efficient Concurrency Control on Modern Hardware“ von Wang et al. [7] vorgestellte Verfahren zur Sicherstellung der Serialisierbarkeit von Transaktionsplänen erläutert. Dabei werden bestehende Verfahren für das Verwalten nebenläufiger Zugriffe auf eine Datenbank, allgemein als Concurrency-Control(CC) bekannt, so erweitert, dass diese die Serialisierbarkeit der entstehenden Pläne gewährleisten. Untersuchte Verfahren sind dabei beispielsweise Snapshot-Isolation(SI), Read-Committed(RC) oder Read-Committed mit Locks(RCL).

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.

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. [8] 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-Concurreny-Control-Verfahren auf. In einem System, welches Multiversion-Concurreny-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
- $T \xleftarrow{r:w} T_j$ **Anti-Abhängigkeit:** T liest eine Version, die T_j überschrieben hat, weshalb T vor T_j serialisiert werden muss

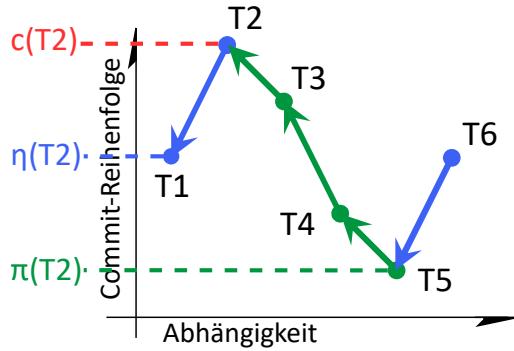


Abbildung 2: Veranschaulichung von Back- und Forward-Edges sowie den Zeitstempeln von Transaktion T2

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 T_2 zu T_5 .

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 T_2 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 $U \xleftarrow{b} T$ lässt sich feststellen, wenn ein Vorgänger U gefunden wird, sodass die folgende Ungleichung erfüllt ist.

$$\pi(T) \leq c(U) \leq c(T) \quad (1)$$

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 T_2 eine solche Verletzung des Ausschlussfensters vorliegt, da die Transaktion T_1 , welche ein Vorgänger von T_2 ist, zwischen $c(T_2)$ und $\pi(T_2)$ committet wurde. Dies wird außerdem klar dadurch, dass die Transaktion T_6 in diesem Beispiel ebenfalls die Transaktion T_1 sein könnte, wovon T_2 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.

$$\pi(T) \leq \eta(T) \quad (2)$$

Wird diese Bedingung auf das in Abbildung 2 beschriebene Beispiel angewendet, wird deutlich, dass die Ungleichung für Transaktion T_2 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 dieselbe 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 Ungleichung 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.

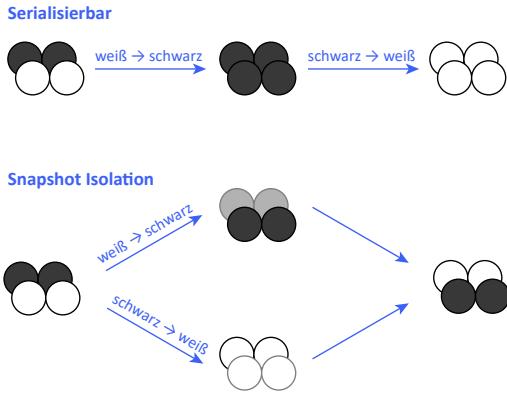


Abbildung 3: Beispiel für das Problem des Write Skew bei der Snapshot Isolation [4]

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. All diese Systeme wurden von den Autoren in das Online-Transaction-Processing(OLTP)-System Silo eingebaut und auf dieser Basis verglichen. 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 committed 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 manipulierte hat, welches T ebenfalls manipulierte hat.

Das Verfahren lässt sich nicht wie beispielsweise Read Committed 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 einem entsprechenden 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 würde, woraufhin jede Transaktion die für sie relevante Hälfte der Kugeln umfärbt 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 vorhergehenden vorgestellten Problematik bei der Snapshot-Isolation verhindert. Somit sind bei der Verwendung dieses Verfahrens keinerlei Abhängigkeitszyklen möglich.

Optimistic-Concurrency-Control (OCC): Zu guter Letzt soll noch das üblicherweise in Silo verwendete CC-Verfahren untersucht werden, welches eine Variante der Optimistic-Concurrency-Control darstellt, weshalb es im Folgenden als OCC bezeichnet wird. Dieses Verfahren sichert ebenfalls die Serialisierbarkeit des Plans und verspricht einen hohen Durchsatz sowie gute Skalierbarkeit. Weitere Informationen zu Optimistic-Concurrency-Control und der in Silo implementierten Variante finden sich in [5] und [6].

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. Obwohl in dem Artikel, auf den sich diese Ausarbeitung bezieht [7] lediglich der TPC-C Benchmark betrachtet wird, ist hier anzumerken, dass die Autoren in ihrem weiterführenden Artikel [8] außerdem den TPC-CC und TPC-EH Benchmark verwendet haben um die Ergebnisse zu bestätigen.

3.3 Testsystem

Der Vollständigkeit halber sei hier noch erwähnt, dass zum Durchführen der Tests ein Server mit vier Sockeln verwendet wurde, der mit vier Intel Xeon E7-4807 Prozessoren bestückt war und somit über insgesamt 24 physikalische Rechenkerne verfügte. Außerdem war das System mit 64GB Hauptspeicher ausgestattet, wodurch es

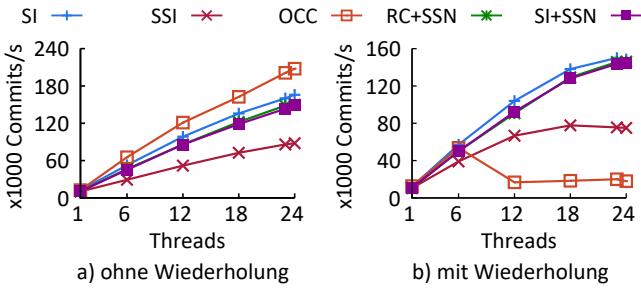


Abbildung 4: Durchsatz der Payment-Transaktion des TPC-C Benchmarks

sich um einen mittelgroßen Gerät handelt, wie man es beispielsweise in einem mittelständischen Unternehmen finden würde.

4 EVALUATION

Zur Bewertung des vorgestellten Verfahrens wird besonderes Augenmerk auf schreibintensive Transaktionen, die Performance unterschiedlicher Transaktionstypen und Commit- und Abbruchraten gelegt. Diese Eigenschaften werden wie zuvor beschrieben mit dem TPC-C Benchmark beobachtet.

Schreibintensive Transaktionen: Die im TPC-C Benchmark enthaltene Payment-Transaktion stellt eine besonders schreibintensive Anwendung dar, weshalb sie für diesen Test verwendet wird. Dabei wurde zwar der gesamte TPC-C Mix ausgeführt, allerdings wurde nur die besagte Payment-Transaktion beobachtet.

Für eine differenzierte Betrachtung wurden zwei verschiedene Testszenarien beobachtet, wobei zum einen sämtliche Transaktionen, die abgebrochen werden mussten ohne eine Wiederholung fallen gelassen wurden. Zum anderen wurde in einem Test die Performance des Systems beobachtet wenn die fehlgeschlagenen Transaktionen wiederholt werden bis diese abgeschlossen werden können. Die Ergebnisse dieser Auswertung sind in Abbildung 4 grafisch dargestellt.

Dabei fällt auf, dass die Verwendung von Read-Committed in Verbindung mit dem Serial Safety Net einen Durchsatz leistet, der fast doppelt so groß ist wie der der Serializable-Snapshot-Isolation. Der Grund dafür liegt laut den Autoren darin, dass der Hauptgrund für Transaktionsabbrüche bei der Verwendung von SSI in dem sogenannten Temporal Skew liegt. Dabei versucht eine Transaktion eine Version zu überschreiben, welche nach ihrem Snapshot erstellt wurde, was bei SSI nicht zulässig ist. RC hat damit kein Problem, da jederzeit auf die neueste Version zugegriffen wird, was sich auch durch die Erweiterung durch das SSN nicht ändert.

Außerdem ist zu erkennen, dass auch die Snapshot-Isolation in Verbindung mit dem SSN eine hervorragende Performance in beiden Testfällen besitzt.

Die Verwendung der Optimistic-Concurrency-Control zeigt bei dem Test ohne Wiederholungen die beste Performance, da ein Großteil des Verwaltungsaufwandes der anderen CC-Verfahren entfällt. Wird allerdings verlangt, dass die Transaktionen nach einem Abbruch wiederholt werden müssen, so sinkt der Durchsatz der OCC weit unter den Durchsatz der anderen CC-Verfahren, da die Zahl

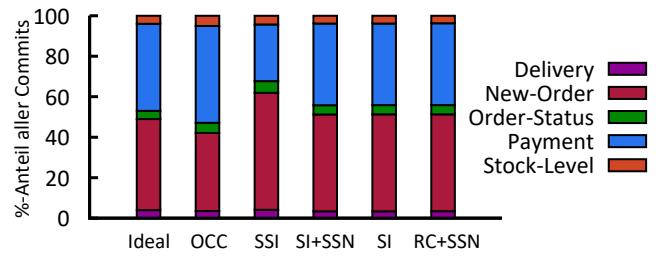


Abbildung 5: Anteil des Durchsatzes der verschiedenen Transaktionstypen des TPC-C Benchmarks

der zu wiederholenden Transaktionen so hoch ist, dass bei stark nebenläufigen Transaktionen die Performance einbricht.

Performanz unterschiedlicher Transaktionsarten: Zum Testen des Durchsatzes der verschiedenen Transaktionstypen des TPC-C Benchmarks wurden die vorher untersuchten Verfahren mit dem aus der TPC-C Spezifikation entnommenen Idealwert verglichen. Der Test wurde auf einer Maschine mit 24 Threads und ohne Wiederholen von abgebrochenen Transaktionen durchgeführt, wodurch das in Abbildung 5 dargestellte Ergebnis erzielt wurde.

Die Grafik zeigt den Anteil der erfolgreich abgeschlossenen Transaktionen jedes Typs an der Gesamtzahl der erfolgreich abgeschlossenen Transaktionen. Besonderes Augenmerk fällt hierbei auf die Payment-Transaktion, welche einen besonders schreibintensiven Anwendungsfall abbildet.

Hier liefert die OCC einen sehr hohen Durchsatz, was darauf schließen lässt, dass das ursprünglich in Silo implementierte System besonders gut für schreiblastige Applikationen geeignet ist, nicht jedoch für Transaktionen, die zusätzlich viele Leseoperationen durchführen. Außerdem fällt auf, dass die SSI eine sehr schlechte Leistung für schreibintensive Anwendungen liefert, wodurch grundsätzlich die Ergebnisse aus dem vorhergehenden Abschnitt bestätigt werden. Bemerkenswert ist, dass die Erweiterung der SI durch das SSN keine nennenswerte Änderung an der dargestellten Verteilung bewirkt, wodurch erkennbar ist, dass das SSN einen gleichmäßigen Einfluss auf die Performance der verschiedenen Transaktionstypen hat und nicht eine besondere Transaktionsart wie beispielsweise leseintensive Transaktionstypen besonders beeinflusst. Ebenfalls vielversprechend ist die Tatsache, dass beide Varianten des SSN sehr nah an der durch die TPC-C Spezifikation vorgesehenen Idealverteilung liegen, wodurch sichergestellt ist, dass es sich dabei um ein ausgewogenes Verfahren handelt, mit dem eine Vielzahl von Anwendungsfällen effizient umgesetzt werden kann.

Commit- und Abbruchraten: Für die Untersuchung der Commit- und Abbruchraten wurde wieder der gesamte TPC-C Mix beobachtet, wobei die Untersuchungen jeweils mit und ohne Wiederholung fehlgeschlagener Transaktionen durchgeführt wurden.

Die dazu in Abbildung 6 dargestellten Ergebnisse geben dazu Auskunft über die diesbezüglichen Eigenschaften der unterschiedlichen Verfahren. Die in a und c dargestellten Commitraten bestätigen die Ergebnisse, die zuvor bei der Untersuchung der Payment-Transaktion festgestellt wurden, da sich der gesamte TPC-C Mix sehr ähnlich wie diese verhält. In a und b dargestellten Ergebnisse zeigen, dass alle Verfahren gut mit der Anzahl der Threads skalieren,

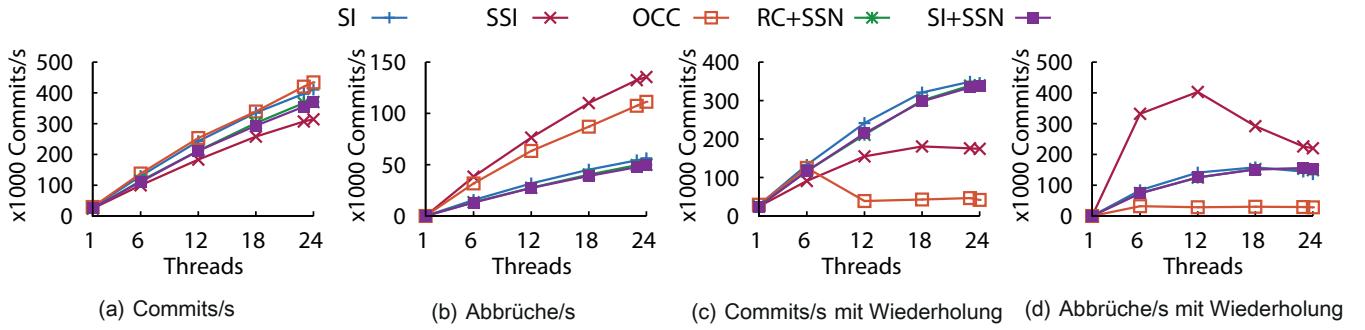


Abbildung 6: Commit- und Abbruchraten des TPC-C Benchmarks mit und ohne Wiederholung abgebrochener Transaktionen

wenn fehlgeschlagene Transaktionen nicht wiederholt werden. Außerdem lässt sich in b erkennen, dass SSI und OCC fast doppelt so oft Transaktionen abbrechen wie die Varianten des SSN. Aufgrund der kleinen Grafik lässt es sich zwar schwer erkennen, allerdings schneidet RC+SSN in diesem Fall minimal besser ab als SI+SSN, da dieses einige Transaktionspläne erlaubt, welche von letzterem verboten werden.

In Teil d der Abbildung fällt auf, dass SSI eine weitaus höhere Abbruchrate als die anderen Verfahren besitzt. Zunächst verwundern dürfte außerdem die Tatsache, dass OCC in dieser Darstellung die geringste Abbruchrate besitzt, was allerdings darin begründet ist, dass das in Silo eingesetzte Verfahren einen hohen Overhead beim Einfügen von Indexeinträgen verursacht wodurch nur wenige Transaktionen pro Sekunde betrachtet werden, was sich beispielsweise auch in c widerspiegelt.

Zulässige Transaktionspläne: Die Unterschiede zwischen den untersuchten Verfahren zur Sicherung der Serialisierbarkeit werden besonders deutlich, wenn ein paar beispielhafte Abhängigkeitstrukturen in Transaktionsplänen verglichen werden. Dazu sind in Abbildung 7 einige Transaktionspläne aufgezeigt und dazu vermerkt, ob jeweils das SSN, 2PL oder SSI dies Pläne akzeptiert oder verwirft. Dabei kann angenommen werden, dass in den entsprechenden Graphen keine Abhängigkeitszyklen enthalten sind und in einem perfekten System somit keiner der Pläne verworfen werden müsste.

Es wird direkt deutlich, dass SSN wesentlich mehr Pläne zulässt als die anderen Verfahren, es verwirft sogar nur den Plan f, da dieser als einziger die Ungleichung 2 aus Kapitel 2 erfüllt und somit nicht zulässig ist.

Das 2PL verbietet dagegen sämtliche Pläne abgesehen von Plan a, da dieses Verfahren das Vorkommen von Back-Edges vollständig verbietet und alle anderen Pläne solche Abhängigkeiten enthalten.

Die SSI erlaubt grundsätzlich die Pläne a und b und verbietet in jedem Fall den Plan d. Ein Fall wie ihn Plan c zeigt, wird nur von SSI akzeptiert, wenn die Transaktion ganz links ausschließlich Leseoperationen enthält und schon ausreichend alt ist. Der Transaktionsplan e kann bei der Verwendung von SI erst gar nicht auftreten und ist somit auch bei der Verwendung von SSI unmöglich. Grundsätzlich wird der Plan f zwar akzeptiert, ist aber die Transaktion T mit seinem Vorgänger durch eine harmlose Anti-Abhängigkeit verbunden, so wird auch dieser Plan von SSI verworfen.

Somit sollte deutlich werden, dass das SSN viele zulässige Transaktionspläne erlaubt, die die anderen Verfahren unnötigerweise verwerfen, wodurch ebenfalls ein deutlicher Performanzgewinn begründet werden kann.

5 FAZIT

Das Verfahren „The Serial Safety Net“ verwendet hoch performante Concurrency-Control-Verfahren und stellt für diese die Serialisierbarkeit sicher. Dies geschieht ohne einen besonders großen Overhead zu verursachen und führt damit zu einer hohen Performanz, wobei allerdings keinerlei Anomalien abgesehen von Phantomen entstehen können. Die Effektivität und Skalierbarkeit in realen Systemen wurde mithilfe des TPC-C Benchmarks verifiziert, wodurch es für moderne OLTP-Anwendungen bestens geeignet sein sollte. Außerdem ist das Verfahren unkompliziert zu implementieren, worauf in [7] genauer eingegangen wird, und somit wenig anfällig für Fehler, weshalb es als ein sicheres Verfahren gelten kann.

Das Serial Safety Net bietet insgesamt weniger Transaktionsabbrüche, eine höhere Robustheit gegenüber dem Wiederholen von Transaktionen und ein besseres Verhalten für schreibintensive Aufgabenbereiche als die verglichenen Verfahren, wodurch einem Einsatz in realen Systemen nichts mehr im Weg steht.

6 PERSÖNLICHE MEINUNG

Um einen besseren Eindruck von dem bearbeiteten Artikel zu vermitteln, soll schlussendlich noch meine persönliche Meinung zu der Vorgehensweise der Autoren genannt werden. Allgemein finde ich, dass der Text einsteigerfreundlich geschrieben ist, sodass er auch für Leser, die nicht besonders tief mit der Materie vertraut sind, leicht verständlich sein sollte. Abgesehen davon werden aber auch nicht zu viele Grundlagen aus dem Themenbereich wiederholt, sodass sich auch erfahrene Leser nicht langweilen dürften.

Außerdem wirkt der Text allgemein wissenschaftlich fundiert, wozu auch der im Anhang des Originals stehende Beweis beiträgt. Besonders überzeugend ist auch, dass die Autoren in dem nachfolgend veröffentlichten Artikel [8] weitere Tests in unterschiedlichen Szenarien durchführen, welche die Ergebnisse weiter untermauern.

Weniger gut gefällt mir, dass die Autoren einige für mich offensichtliche Auffälligkeiten teilweise überhaupt nicht erklären. Beispielsweise wäre hier zu nennen, dass in Abbildung 6 d die Transaktionsabbrüche bei Verwendung der SSI mit zunehmender

	(a)	(b)	(c)	(d)	(e)	(f)
SSN	Erlaubt	Erlaubt	Erlaubt	Erlaubt	Erlaubt	Verboten
2PL	Erlaubt	Verboten	Verboten	Verboten	Verboten	Verboten
SSI	Erlaubt	Erlaubt	Bedingt	Verboten	Unmöglich	Bedingt

Abbildung 7: Einige Beispiele für mögliche Transaktionspläne und ob diese von SSN, 2PL oder SSI akzeptiert werden

Thread-Zahl rasant steigt, ab 18 Threads aber wieder langsam abnimmt. Dieses Verhalten kann ich mir absolut nicht erklären und auch im Artikel wird darauf nicht weiter eingegangen.

Des weiteren wird in dem Artikel nicht wirklich eine klare Begründung für die schlechte Performanz der Serializable-Snapshot-Isolation geliefert. Es wird zwar an einer Stelle gesagt, dass es etwas mit einer hohen Zahl von Indexoperationen zutun hat, allerdings wird darauf nicht weiter eingegangen. In dem später erschienenen Artikel [8] werden hier zwar einige weitere Erklärungen gegeben, allerdings fehlen dort meiner Meinung nach einige Hinweise in dem Originalartikel [7].

Als weitere Begründung für die schlechte Performanz von SSI wird genannt, dass eine hohe Zahl von Transaktionsabbrüchen durch den sogenannten Temporal Skew verursacht werden. Dabei schreibt eine Transaktion auf einen Wert, der nach seinem Snapshot überschrieben wurde. Dieser Fall tritt allerdings bei der normalen SI ebenfalls auf, weshalb dies nicht erklärt, warum SI+SSN an den meisten Stellen deutlich besser abschneidet als die SSI.

Bei der Erklärung von Abbildung 6 d wurde vorher begründet, dass SSI eine besonders geringe Abbruchrate besitzt, da die allgemeine Performanz so schlecht ist, dass nur wenige Transaktionen pro Sekunde bearbeitet werden, was natürlich zu einer geringen Zahl von Transaktionsabbrüchen führt. Der gleiche Effekt müsste aber doch auch bei den anderen Verfahren in irgendeiner Form existieren, weil diese ja auch alle einen unterschiedlichen Overhead produzieren, wodurch die Aussagekraft der Ergebnisse, die in Abbildung 6 dargestellt wurden, meiner Meinung nach etwas eingeschränkt ist.

LITERATUR

- [1] [n. d.]. TPC-C is an On-Line Transaction Processing Benchmark. ([n. d.]).
- [2] Atul Adya, Barbara Liskov, and Patrick O’Neil. 2000. Generalized isolation level definitions. In *Proceedings of 16th International Conference on Data Engineering (Cat. No.00CB37073)*. <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://doi.org/10.1145/223784.223785>
- [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] H. T. Kung and John T. Robinson. 1981. On Optimistic Methods for Concurrency Control. *ACM Trans. Database Syst.* (1981). <https://doi.org/10.1145/319566.319567>
- [6] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy Transactions in Multicore In-memory Databases. In *Proceedings*

of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM. <https://doi.org/10.1145/2517349.2522713>

- [7] 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*. ACM. <https://doi.org/10.1145/2771937.2771949>
- [8] Tianzheng Wang, Ryan Johnson, Alan Fekete, and Ippokratis Pandis. 2016. Efficiently making (almost) any concurrency control mechanism serializable. (2016). <https://doi.org/10.1007/s00778-017-0463-8>