AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

11. Übungsblatt

Für die Übung am Donnerstag, **25. Januar 2007**,
von 15:30 bis 17:00 Uhr in 13/222.

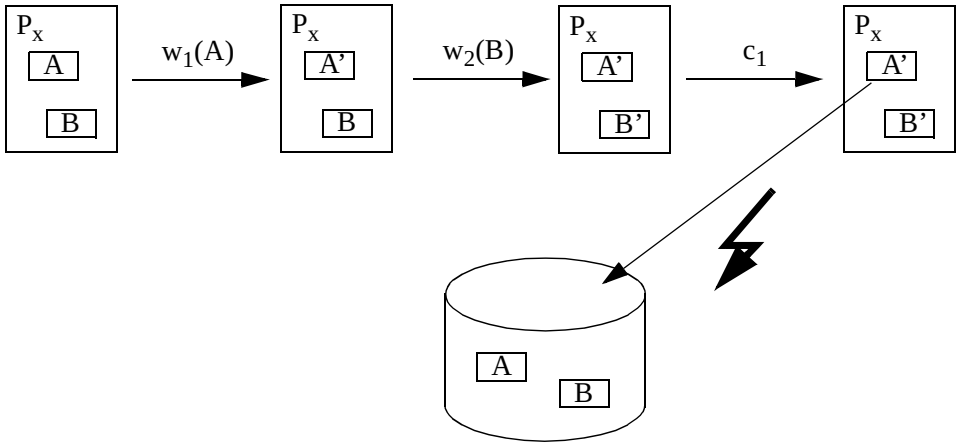
Aufgabe 1: Ausschreib- und Seitenersetzungsstrategie

Demonstrieren Sie anhand eines Beispiels, dass man die Ausschreibstrategie FORCE nicht mit der Seitenersetzungsstrategie NOSTEAL kombinieren kann, wenn parallele Transaktionen gleichzeitig Änderungen an Datensätzen innerhalb einer Seite durchführen. Betrachten Sie dazu zwei Transaktionen T_1 und T_2, die parallel ablaufen und die Datensätze $A(T_1)$ resp. $B(T_2)$ der Seite P modifizieren.

Lösung:

Hinweis: Die Kombination der Ausschreibstrategie FORCE (erzwungenes Ausschreiben aller Änderungen einer TA bei ihrem Commit) mit der Seitenersetzungsstrategie NOSTEAL (kein Verdrängen "schmutziger" Seiten) ist prinzipiell nur mit einer atomaren Einbringstrategie korrekt möglich.

Wenn bei der Verwendung von Satzsperren zwei Transaktionen gleichzeitig auf zwei verschiedenen Objekten innerhalb einer Seite arbeiten, kommt es zu der Situation, dass die Transaktion (in diesem Beispiel T_1), die zuerst ihr Commit durchführt, ihre Änderungen entsprechend FORCE zum Commit-Zeitpunkt einbringen muss. Gleichzeititg verbietet NOSTEAL das Einbringen der noch nicht festgeschriebenen Änderungen von Transaktion T_2. Aus diesem Grund erfordert die Kombination von FORCE und NOSTEAL zwingend Seitensperren, wodurch diese Situation verhindert wird.



Aufgabe 2: Charakterisierung von Sicherungspunkt-Schemata

Zur Unterstützung des Entwurfs von Logging- und Recovery-Verfahren wurden verschiedene Sicherungspunkt-Schemata eingeführt:

- transaktionsorientierte Sicherungspunkte (TOC)
- transaktionskonsistente Sicherungspunkte (TCC)
- aktionskonsistente Sicherungspunkte (ACC)
- Fuzzy Checkpoints

Als Einbringstrategien wurden

- das direkte Einbringen (NON-ATOMIC) und
- das verzögerte Einbringen zu kontrollierten Zeitpunkten (ATOMIC)

unterschieden.

Als Seitenersetzungsstrategien kommen

- ein Erlauben des Verdrängens schmutziger Seiten (STEAL)
- ein Verbot des Verdrängens schmutziger Seiten (NOSTEAL)

in Frage.

Die vier Möglichkeiten der Sicherungspunkterzeugung können je mit den beiden Einbringstrategien und mit den beiden Seitenersetzungsstrategien kombiniert werden, sodass für Logging- und Recovery-Strategien zunächst 16 verschiedene Ausgangssituationen entstehen, von denen einige jedoch nicht möglich sind. Der Zustand der Datenbank im Fehlerfall und die Qualität des Sicherungspunktes sind wichtige Parameter für den Entwurf von Logging- und Recovery-Verfahren.

Beschreiben Sie zunächst zur Wiederholung für jedes Sicherungspunkt-Schema die Aktivitäten beim Anlegen des Sicherungspunktes und charakterisieren Sie die jede der 16 Kombinationen nach den im folgenden genannten Kriterien. Überprüfen Sie aber zunächst die Sinnhaftigkeit der jeweiligen Kombination. Falls die Kombination nicht umsetzbar ist oder umsetzbar aber nicht sinnvoll ist, begründen Sie dies.

a) Qualität der persistenten DB im Sicherungspunkt (d.h. bei dessen erfolgreichem Abschluss)
   Vorschläge für Qualitätsstufen:
   - transaktionskonsistent: die DB enthält alle Änderungen erfolgreich abgeschlossener Transaktionen und keine Änderungen nicht erfolgreich abgeschlossener Transaktionen
   - aktionskonsistent: die DB enthält die Änderungen jeder Aktion (Schreiboperationen eines Transaktionsprogramms) vollständig oder gar nicht. Diese Aktionen können bereits festgeschrieben (erfolgreiches Commit der TA) oder schmutzig sein.
   - chaotisch: die DB enthält schmutzige wie festgeschriebene Änderungen. Auswirkungen einzelner Aktionen sind möglicherweise nur teilweise in der Datenbank repräsentiert. So kann beispielsweise eine Aktion, die zum Einfügen eines Satzes in eine Seite geführt hat, bereits in einer Indexseite in der DB repräsentiert sein, aber noch nicht die Änderung in der Datenseite.

   Wenn diese Qualitätsstufen nicht ausreichen, um die Qualität eines SP zu charakterisieren, führen Sie ggf. weitere Stufen ein.

b) den Zustand der persistenten DB nach einem Systemfehler:
   Welche Qualität hat die Datenbank bei einem Systemfehler. Verwenden Sie auch hier die Qualitätsstufen von a). Wenn sich die Qualität je nach Zeitpunkt (direkt vor, direkt nach, während dem Anlegen eines Sicherungspunktes) des Systemfehlers unterscheidet, beschreiben Sie den Zustand für die ungünstigste Konstellation, da diese die Situation ist, von der bei der Recovery auszuegehen ist.

c) den Zustand des Sicherungspunkts in einem SP-Intervall:
Wenn der Sicherungspunkt die in a) erreichte Qualität im Anschluss beibehält, nennt man ihn beständig oder stabil. Verändert sich die Qualität des Sicherungspunktes (auch innerhalb einer "Qualitätsstufe"), nennt man ihn transient.

d) die Voraussetzungen bezüglich des Sperrens (z.B. Einschränkungen des Sperrgranulats):
Sind Satzsperren möglich, oder erfordert das Verfahren zwingend ein größeres Sperrgranulat?

e) die Grenzen der REDO- und UNDO-Recovery:
Sicherungspunkte dienen dazu, beim Wiederanlauf nicht alle Änderungen seit Systemstart auf Redo bzw. Undo überprüfen zu müssen. Ab welcher Position muss das Log in der Redo-Phase gelesen werden? Bis zu welcher Log-Position erstreckt sich die Undo-Phase.

Lösung:

Wiederholung: ATOMIC vs. NONATOMIC

In der Vorlesung wurden zwei Einbringstrategien unterschieden:

NONATOMIC bedeutet, dass jede Seite einer einzigen Position auf dem Externspeicher zugeordnet ist. Mit dem Ausschreiben einer solchen Seite wird daher unweigerlich ihr vorheriger Zustand auf dem Externspeicher überschrieben und ist daher gleichzeitig ein Einbringen. Folglich ist beim Einbringen schmutziger (=nicht festgeschriebener, nicht "committeter") Änderungen sicherzustellen, dass die Änderung rückgängig gemacht werden kann. Dazu wird gemäß dem "Write-ahead-log"-Prinzip (WAL) zuerst Log-Information mit Undo-/Redo-Informationen geschrieben, um die Änderungen rückgängig machen zu können. Da bei NONATOMIC nicht nur das Ausschreiben einer Seite selbst unterbrochen werden kann, sondern insbesondere auch das Ausschreiben mehrerer Seiten ein durch Systemfehler unterbrechbarer, nicht-atomarer Vorgang ist, können Änderungen einer Datenbankoperation, die oft mehrere Seiten betreffen, nicht garantiert vollständig eingebracht werden. Dies hat zur Folge, dass der Zustand der Datenbank im Falle eines Systemfehlers im Allgemeinen weder transaktions- noch operationskonsistent ist.

ATOMIC bedeutet, dass eine beliebige Anzahl von Seiten durch atomares "Umschalten" zeitgleich eingebracht werden können. Dies wird dadurch umgesetzt, dass im Puffer geändert Seiten, die ausgeschrieben werden sollen, nicht auf ihren ursprünglichen Platz geschrieben werden, sondern auf einen bisher ungenutzten Platz. Die Seiten werden über eine Seitentabelle aufgefunden. Eine laufende Seitentabelle, die (aus Performancegründen) zumindest teilweise im Hauptspeicher liegt, zeigt auf die Seiten mit den aktuellen Änderungen, während die alte Seitentabelle auf dem persistenten Speicher unverändert bleibt und auf die alten Seiten zeigt. Beim Einbringen der Änderungen durch "Umschalten" werden nun die laufende Seitentabelle (ebenfalls auf eine andere Stelle, NICHT über die alte Seitentabelle) und alle evtl. noch im DB-Puffer geändert vorliegenden Seiten ausgeschrieben. Erst wenn dieser Ausschreibvorgang vollständig abgeschlossen ist, wird durch Setzen eines Switch-Bits (was atomar möglich ist) auf dem persistenten Speicher von der alten auf die neue Seitentabelle umgeschaltet, womit nun alle Änderungen Teil der persistenten DB werden und auch im Falle eines Systemfehlers sichtbar sind.

Wiederholung: STEAL vs. NOSTEAL:

STEAL bedeutet, dass schmutzige Seiten, also Seiten mit Änderungen, die von der ändernden Transaktion noch nicht festgeschrieben ("committet") wurden, aus dem DB-Puffer verdrängt und auf den persistenten Speicher geschrieben werden dürfen. In Kombination mit ATOMIC ist dieses Ausschreiben ohne Einbringen in die persistente DB möglich. Bei NONATOMIC ist ein Ausschreiben beim Verdrängen immer auch ein Einbringen in die DB.

NOSTEAL verbietet das Verdrängen schmutziger Seiten aus dem Puffer.

Transaktionsorientierte Sicherungspunkte (TOC):

Die Ausschreibstrategie FORCE (alle Änderungen einer TA werden spätestens bei ihrem Commit in die persistente DB eingbracht) kann als ein spezieller Sicherungspunkttyp aufgefasst werden. Der Sicherungspunkt bezieht sich dabei auf genau eine Transaktion.

ATOMIC, NOSTEAL

Die Verwendung einer atomaren Einbringstrategie in Kombination mit FORCE/TOC ist konzeptionell so zu verstehen, dass pro Transaktion zumindest logisch eine eigene Seitentabelle vorgehalten wird, welche Zeiger auf die von dieser Transaktion geänderten Seiten enthält. Beim Commit, also beim Sicherungspunkt, werden die geänderten Seiten ausgeschrieben (noch als Schattenseiten), anschließend die TA-spezifische Seitentabelle (bzw. der Teil der gesamten Seitentabelle der für die ihr Commit ausführnde TA), dann wird die Seitentabelle atomar eingebracht durch "Umschalten" und dabei für alle späteren TAen sichtbar.
Da nebenläufige, noch aktive Transaktionen keine ihrer Änderungen eingebracht haben (atomares Umschalten und NOSTEAL), sind beim SP nur die Änderungen der das Commit ausführenden Transaktion (und natürlich aller vorher erfolgreich abgeschlossenen TAen) in der DB enthalten, daher ist **a) die Qualität des SP transaktionskonsistent** und behält diese Qualität auch bei einem Systemfehler, **b) der Zustand der DB nach einem Systemfehler ebenfalls transaktionskonsistent**. Am Inhalt der persistenten DB ändert sich bis zum nächsten SP (Commit der nächsten TA) nichts, somit ist **c) der Zustand des SP in einem SP-Intervall beständig**. Um NOSTEAL sicherzustellen, dürfen Transaktionen nicht parallel in Datensätzen in der gleichen Seite Änderungen vornehmen, dementsprechend sind **d) als Voraussetzung bezüglich des Sperrens Seitensperren** erforderlich. Da alle Änderungen erfolgreicher TA vollständig eingebracht wurden und nicht abgeschlossene TAs (wegen NOSTEAL) keine schmutzigen Änderungen eingebracht haben ist **e) weder REDO noch UNDO erforderlich**.

ATOMIC, STEAL:

Wird bei TOC/FORCE atomares Einbringen mit STEAL kombiniert, bedeutet dies lediglich, dass auch schmutzige Seiten aus dem DB-Puffer verdrängt werden dürfen. Bei ATOMIC ist dies möglich, OHNE die schmutzigen Seite einzubringen (sie landet in einem zuvor leeren Bereich des persistenten Speichers und werden nicht auf der aktuellen festgeschriebenen (Schatten-)Seitentabelle, sondern nur auf der noch nicht festgeschriebenen neuen Seitentabelle (u.U. im Hauptspeicher) "verzeigert". Dadurch ist wie bei NOSTEAL die **a) Qualität des Sicherungspunktes transaktionskonsistent**, da auch hier erst beim Umschalten die Seiten eingebracht werden (die damit nicht mehr schmutzig sind). Folglich ist er **b) auch nach einem Systemfehler transaktionskonsistent**. Da sich der Zustand des Sicherungspunktes bis zum nächsten Sicherungspunkt nicht ändert, ist er **c) innerhalb eines SP-Intervalls beständig**. Auch hier sind wie bei NOSTEAL **d) Seitensperren erforderlich** da andernfalls auch hier das Commit einer TA (der SP) parallele Änderungen einer noch laufenden Transaktion einbringen würde. Wie zuvor ist **e) weder REDO noch UNDO erforderlich**.

NON-ATOMIC, NOSTEAL:

Die Kombination von TOC/FORCE mit einem nicht-atomaren Einbringverfahren und NOSTEAL ist nicht möglich, da TOC/FORCE erfordert, dass bei Commit alle Änderungen eingebracht sind (das Auschreiben kann nicht atomar erfolgen, muss daher als Folge unterbrechbarer Operationen vor dem Commit passieren). Gleichzeitig verbietet NOSTEAL das Einbringen noch nicht festgeschriebener (nicht "committeter") Änderungen.

NON-ATOMIC, STEAL:

Mit dem Anmelden des Sicherungspunktes (= Commit-Wunsch des Transaktionsprogramms) werden alle noch nicht ausgeschriebenen Änderungen der TA eingebracht (was dank STEAL erlaubt ist).
Die **a) Qualität des Sicherungspunktes** ist daher (bezogen auf die gesamte Datenbank) **chaotisch**, da parallel schmutzige Seiten anderer Transaktionen eingebracht werden können, folglich ist auch der **b) Zustand der DB nach einem Systemfehler chaotisch**. nicht besser. Der (sowieso chaotische und damit für logisches Logging nicht nutzbare) DB-Zustand ist zudem wegen STEAL und dem so erlaubte Verdrängen schmutziger Seiten durch den Pufferverwalter in stetem Fluß und daher **c) innerhalb eines SP-Intervalls transient**. Es gibt d) keine Einschränkungen bzgl. des Sperrgranulats. Ein **e) REDO ist nicht erforderlich**, da weiterhin sichergestellt wird, das erst nach dem Ausschreiben aller Änderungen einer TA das erfolgreiche Commit an die Anwendung gemeldet wird. Wegen der Verdrängung schmutziger Seiten durch STEAL ist dagegen ein **UNDO erforderlich**, dass **bis zum BOT der ältesten beim Crash aktiven TA** reicht.

Transaktionskonsistente Sicherungspunkte (TCC):

Bei transaktionskonsistenten Sicherungspunkten werden alle nach Anmelden des Sicherungspunktes (durch das DBMS) neu eintreffenden Änderungstransaktionen bis zum Abschluss des SP verzögert. Alle noch laufenden Änderungs-TA können ihre Arbeit beenden (was einen vorher nicht zu bestimmenden und u.U. sehr langen Zeitraum mit Totzeit für alle neuen Änderungs-TAen bedeutet). Sind alle Änderungstransaktionen abgeschlossen, werden alle noch nicht eingebrachten Änderungen im DB-Puffer ausgeschrieben (da keine Änderungs-TA mehr aktiv ist, ist sichergestellt dass alle Änderungen NON-dirty, also festgeschrieben oder "committet" sind. Nach dem Ausschreiben des Puffers wird der Normalbetrieb fortgesetzt, wartende Änderungs-TAen können ihre Arbeit aufnehmen.

ATOMIC, NOSTEAL

Bei ATOMIC/NOSTEAL werden im SP alle Änderungen zunächst als Schattenseiten ausgeschrieben, dann die neue Seitentabelle, die schließlich atomar durch Umschalten gültig gemacht wird. Somit ist sichergestellt, dass **a) die Qualität des SP TA-konsistent ist**. Da weiteres Einbringen nur in Sicherungspunkten passiert, bleibt **b) die DB beim Systemfehler TA-konsistent**, die **c) Qualität bleibt also zwischen zwei SPen stabil**. Es gibt **d) keine Restriktionen bzgl. des Sperrgranulats**, da das Einbringen von parallelen Änderungen in einer Seite ausschließlich im Sicherungspunkt passiert, also zu einem Zeitpunkt, wo die nebenläufigen TA beide ihre Änderungen bereits eingebracht haben. NOSTEAL wird also auch ohne Seitensperren gewährleistet. Ein **e) REDO beginnt ab dem letzten Sicherungspunkt** (da es keine abgeschlossenen TAen mit nicht eingebrachten Änderungen gibt, die vor dem letzten Sicherungspunkt begonnen haben) und umfasst vollständig alle Operationen aller seit diesem Zeitpunkt "committeten" Transaktionen (da keine ihrer Änderungen seither eingebracht wurde). **Ein UNDO ist nicht erforderlich**, da das Einbringen nur beim Sicherungspunkt geschieht, wenn alle aktiven Änderungs-TAs ihr Commit ausgeführt haben.

ATOMIC, STEAL

Bei der Kombination TCC mit ATOMIC mit STEAL können nun auch zwischen den Sicherungspunkten schmutzige Seiten verdrängt werden. Das Einbringen erfolgt jedoch weiterhin nur im Sicherungspunkt, so dass die **a) Qualität des SP TA-konsistent ist** und auch **b)** beim **Systemfehler TA-konsistent** ist, die **c) Qualität des SP ist somit persistent**. Es gibt **d) keine Einschränkungen bzgl. des Sperrgranulats**. Ein **e) REDO** nach Crash erfolgt wie bei ATOMIC/NOSTEAL für alle TAen, die seit dem letzten Sicherungspunkt ihr Commit durchgeführt haben, beginnend **ab dem letzten SP**. Da die durch STEAL ausgeschriebenen Seiten nicht eingebracht werden, ist **kein Undo** erforderlich.

NON-ATOMIC, NOSTEAL

Bei TCC in Kombination mit nicht-atomarem Einbringen und NOSTEAL ist **a) die Qualität des SP TA-konsistent**, da alle vor der SP-Anmeldung gestarteten Transaktionen im SP bereits abgeschlossen sind und ihre Änderungen einbringen, aber keine neuen parallelen Änderungen möglich sind. Zwar verbietet NOSTEAL ein Verdrängen schmutziger Seiten. Zwischen den Sicherungspunkten können aber jederzeit einzelne (nicht-schmutzige) geänderte Seiten von committeten Transaktionen eingebracht werden (wenn sie nicht parallel von einer noch laufenden TA genutzt werden). Der **b) Zustand des DB bei einem Systemfehler ist** daher **chaotisch** und ständig im Fluß, wodurch **c) die Qualität transient** ist. Es gibt **d) keine Einschränkungen für das Sperrgranulat**. Ein **e) REDO ist ab dem letzten Sicherungspunkt** durchzuführen, da ein erfolgreich abgeschlossener SP sicherstellt, dass alle Änderungen geschrieben wurden. Ein **UNDO ist nicht erforderlich**, da nur festgeschriebene Änderungen ausgeschrieben werden.

NON-ATOMIC, STEAL

Bei dieser Kombination können zwischen Sicherungspunkten nicht nur festgeschriebene, sondern auch schmutzige Änderungen ausgeschrieben und damit eingebracht werden. Die **a) Qualität des SP ist zunächst TA-konsistent**, wegen der Verdrängungsoperationen **beim b) Crash jedoch chaotisch** und somit ist **c) die SP-Qualität transient**. Es gibt **d) keine Einschränkungen bzgl. der Sperren**. **e) REDO ist ab dem letzten SP erforderlich**, **UNDO bis zum BOT der ältesten aktiven Transaktion**.

Aktionskonsistente Sicherungspunkte (ACC):

Bei aktionskonsistenten Sicherungspunkten werden alle bei Anmeldung des SP (durch das DBMS) aktiven Änderungsoperationen zu Ende geführt, neue Änderungen müssen jedoch warten. Sind alle aktiven Änderungen abgeschlossen, werden alle geänderten Seiten ausgeschrieben und eingebracht, anschließend der Normalbetrieb wieder aufgenommen. Da Aktionen deutlich kürzer als Transaktionen sind, ist die Wartezeit bis zum Beginn des Ausschreibens deutlich kürzer als bei TCC, allerdings gilt auch hier, dass das Ausschreiben aller geänderten Seiten sehr lange dauern kann. Da das Einbringen durch den Sicherungspunkt auch während gerade laufender TAen geschieht, werden unweigerlich auch schmutzige Seiten eingebracht, ACC ist daher nicht mit einer NOSTEAL-Einbringungsstrategie verträglich.

ATOMIC, NOSTEAL

nicht möglich.

ATOMIC, STEAL

In Kombination mit STEAL ist ACC nun möglich. Wie zuvor bedeutet STEAL mit ATOMIC kombiniert, das auch außerhalb von Sicherungspunkten schmutzige Seiten ausgeschrieben werden können, aber kein Umschalten der Seitentabellen und damit kein Einbringen dieser schmutzigen Änderungen erfolgt. Der Sicherungspunkt selbst erfolgt durch Ausschreiben der geänderten Seiten und der neuen Seitentabelle und wird durch Umschalten der Seitentabelle abgeschlossen. Die **a) Qualität der DB im Sicherungspunkt is aktionskonsistent**, was bedeutet, das alle durch einzelne Aktionen geänderte Seiten vollständig eingebracht sind. Da außerhalb eines SP schmutziger Seiten zwar ausgeschrieben, aber nicht eingebracht werden, ist der **b) Zustand bei Systemfehlern ebenfalls aktionskonsistent**, folglich ist **c) die Qualität des SP stabil**. Es gibt **d) keine Einschränkungen bzgl. der Sperren**. Ein **e) Redo ist ab dem letzten Sicherungspunkt** erforderlich (da alle vorherigen Änderungen im SP eingebracht wurden), ein **Undo** auch über den Sicherungspunkt hinweg zurück **bis zum BOT der ältesten beim Crash aktiven TA**, da sowohl der SP selbst als auch STEAL schmutzige Seiten in die DB eingebracht haben können. Die beim Sicherungspunkt ausgeschriebene MinLSN (BOT bzw. erste Aktion der ältesten im SP aktiven TA) gibt eine unter Schranke für die Undo-LSN an.

NON-ATOMIC, NOSTEAL

nicht möglich

NON-ATOMIC, STEAL

Für abgeschlossene aktionskonsistente Sicherungspunkte ist **a) die Qualität des SP aktionskonsistent**. Da jedoch jederzeit schmutzige Seiten verdrängt (und wegen NON-ATOMIC auch eingebracht) werden können, ist die **b) Qualität der DB beim Systemfehler** als **chaotisch** anzunehmen, die **c) Qualität des SP ist folglich transient**. Es sind **d) keine Einschränkungen bzgl. des Sperrgranulats** zu beachten. Ein **e) Redo ist ab dem letzten SP erforderlich**, da alle vorherigen Änderungen dort eingebracht wurden, auch die noch nicht festgeschriebenen, weshalb ein **Undo über den SP hinaus bis zum BOT der ältesten beim Systemausfall aktiven TA erforderlich** ist.

Unscharfe Sicherungspunkte (Fuzzy Checkpoints, Fuzzy):

Alle bisherigen Sicherungspunkt-Typen haben das Problem, dass sie ein Ausschreiben aller Änderungen von abschließenden TA (bei TOC) bzw. aller Änderungen im DB-Puffer (TCC, ACC) bewirken. Dadurch wird der Sicherungspunkt-Aufwand enorm und der DB-Puffer wird schlechter genutzt. Insbesondere große Puffer bedeuten nun nicht mehr automatisch eine Verbesserung der DB-Leistung, da sie potentiell mehr geänderte Seiten enthalten und so den SP-Aufwand erhöhen.
Fuzzy Checkpoints verzichten auf das Ausschreiben der Änderungen, die Datenbank wird also nicht in einen aktions- oder gar transaktionskonsistenten Zustand gebracht. Das eigentliche Ziel, die Begrenzung von Undo- und Redo-Aufwand für die Recovery, erfordert dies jedoch nicht. Das Durchführen eines Fuzzy-Checkpoints beschränkt sich auf das Speichern von Statusinformationen im Log: So wird wie bei ACC die Menge der aktiven Transaktionen vermerkt. **Als wichtigste zusätzliche Information speichert der Pufferverwalter für jede geänderte Seite die LSN der ersten Aktion, die nach dem Einlesen der Seite aus der DB die erste Änderung dieser Seite durchgeführt hat**. Das Minimum dieser Werte über alle Seiten, die MinDirtyPageLSN, beschreibt somit die Position im Log, bei der die Redo-Recovery beginnen muss, und wird als Teil der Sicherungspunktinformationen ausgeschrieben. Das eigentliche Ausschreiben geänderter Seiten erfolgt nebenläufig, unabhängig von den Sicherungspunkten. Dies kann nicht nur im Zuge eines Verdrängens der Seite aus dem Puffer passieren, sondern auch in Form einer Synchronisation der Seite mit der DB, wobei die Seite im Puffer verbleibt ("flush"). Erfolgt eine solche Synchronisation wird natürlich die DirtyPageLSN für diese Seite zurückgesetzt (und ggf. die MinDirtyPageLSN als das Minimum über alle Seiten angepasst). Die MinLSN begrenzt wie bei ACC den Undo-Aufwand (als pessimistische Abschätzung), das Minimum aus MinLSN und MinDirtyPageLSN bestimmt den Startpunkt für die Analyse des Logs.

Würde man Fuzzy Checkpoints ohne weitere Vorkehrungen umsetzen, würde jedoch das eigentlich mit SPen zu behebende Problem, nämlich ein Redo seit dem Start des Systems durchführen zu müssen, nicht beseitigt. Seiten mit hoher Zugriffs- und Änderungshäufigkeit ("Hot-Spot-Seiten") würden u.U. nur selten oder niemals verdrängt oder auch nur synchronisiert. Hier muss der Pufferverwalter sicherstellen, dass auch Hot-Spot-Seiten gelegentlich ausgeschrieben werden. Dies kann nach verschiedenen Kriterien geschehen, z.B. dem Erreichen eines Maximalwerts für den Abstand zwischen MinDirtyPageLSN und aktueller LSN, oder durch Einbringen aller Seiten, die schon beim vorherigen Sicherungspunkt geändert im Puffer vorlagen.

ATOMIC, NOSTEAL und STEAL

Die Kombination einer ATOMIC-Einbringstrategie mit Fuzzy Checkpoints ist nicht sinnvoll. Ein Hauptgrund für die Verwendung atomarer Einbring-Strategien ist es ja gerade, durch Sicherstellen eines immer mindestens aktionskonsistenten DB-Zustandes ein logisches Logging-Verfahren einsetzen zu können. Ein Umschalten der Seitentabellen in einem Fuzzy Checkpoint würden jedoch auch nur partielle Änderungen von Aktionen eingebracht und die DB wäre somit chaotisch, der eigentliche Sinn des (erheblichen!) Aufwandes für atomares Einbringen somit verfehlt.

NON-ATOMIC, NOSTEAL

Ein unscharfer Sicherungspunkt erzwingt keinen TA- oder aktionskonsistenten DB-Zustand, die **a) Qualität eines Sicherungspunktes ist chaotisch** und **b) ist daher auch bei einem Systemfehler chaotisch**. Zwar erlaubt NOSTEAL kein Verdrängen von schmutzigen Seiten, es wird jedoch auch kein Einbringen festgeschriebener Änderungen erzwungen. Der Zustand der DB ist zwischen Sicherungspunkten **c) ständigen Änderungen unterworfen, daher also transient**. Es sind d) **keine Einschränkungen bzgl. des Sperrgranulats erforderlich**. Die **e) Redo-Phase beginnt bei MinDirtyPageLSN**, ein **Undo** erfolgt **bis zum BOT der ältesten beim Crash aktiven TA**.

NON-ATOMIC, STEAL

Kombiniert man Fuzzy Checkpoints mit nicht-atomarem Einbringen und einer STEAL-Verdrängungsstratgie, gelten die gleichen Kriterien wie zuvor bei NOSTEAL. Zusätzlich können jetzt auch noch schmutzige Seiten verdrängt werden.

Tabelle:

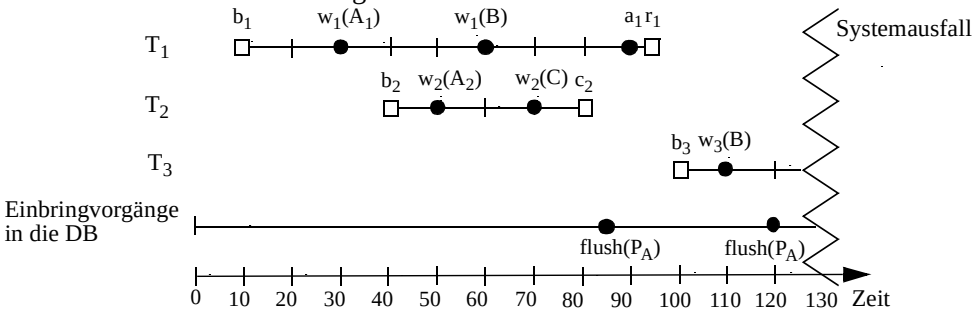
	ATOMIC (Verzögerte Einbringstrategie)		NON-ATOMIC (Direkte Einbringstrategie)	
	NOSTEAL	STEAL	NOSTEAL	STEAL
TOC => FORCE => kein Redo	a) TA-konsistent b) TA-konsistent c) stabil d) Seitensperren e) R: entfällt U: entfällt	a) TA-konsistent b) TA-konsistent c) stabil d) Seitensperren e) R: entfällt U: entfällt	nicht möglich	a) chaotisch b) chaotisch c) transient d) keine e) R: entfällt U: bis BOT ältester akt. TA
TCC => NOFORCE	a) TA-konsistent b) TA-konsistent c) stabil d) keine e) R: ab letztem SP U: entfällt	a) TA-konsistent b) TA-konsistent c) persistent d) keine e) R: ab letztem SP U: entfällt	a) TA-konsistent b) chaotisch c) transient d) keine e) R: ab letztem SP U: entfällt	a) TA-konsistent b) chaotisch c) transient d) keine e) R: ab letztem SP U: bis BOT ältester akt. TA = bis SP
ACC => NOFORCE, STEAL	nicht möglich	a) aktionskonsistent b) aktionskonsistent c) stabil d) keine e) R: ab letztem SP U: bis BOT ältester akt. TA	nicht möglich	a) aktionskonsistent b) chaotisch c) transient d) keine e) R: ab letztem SP U: bis BOT ältester akt. TA
Fuzzy => NOFORCE	Nicht relevant	Nicht relevant	a) chaotisch b) chaotisch c) transient d) keine e) R: ab MinDirtyPageLSN U: bis BOT ältester akt. TA	a) chaotisch b) chaotisch c) transient d) keine e) R: ab MinDirtyPageLSN U: bis BOT ältester akt. TA

Aufgabe 3: Logging und Recovery mit Rollback und Satzsperren

Gegeben sei ein DBMS, das folgende parallel laufende Transaktionen T_1,T_2 und T_3 mit Hilfe von Satzsperren verwaltet. Dabei ändert T_1 die Datenelemente A_1 und B, T_2 die Datenelemente A_2 und C, und T_3 das Datenelement B. Die beiden Datenelemente A_1 und A_2 befinden sich in der Seite P_A, B in P_B und C in P_C.

T_2 wird zum Zeitpunkt 80 erfolgreich beendet (c_2), während T_1 zum Zeitpunkt 90 mit einer Abort-Operation beginnt (a_1). Die Seite P_A wird jeweils zu den Zeitpunkten 85 und 120 aus dem DB-Puffer verdrängt. Alle Rücksetzmaßnahmen von T_1, d. h. die Ausführung ihrer Kompensationsoperationen, sind zum Zeitpunkt 95 (r_1) vor b_3 vollständig ausgeführt worden.

Während der Ausführung der Transaktionen werden keinerlei Sicherungspunkte gesetzt. Bei der Recovery wendet das DBMS vollständiges REDO an.



a) Wie behandelt man die Rollback-Operation?

b) Führen Sie anschließend die in der Abbildung gezeigten Aktionen der Transaktionen nacheinander durch und vervollständigen dabei Sie die folgende Tabelle:

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
...	...	...	...	...	...

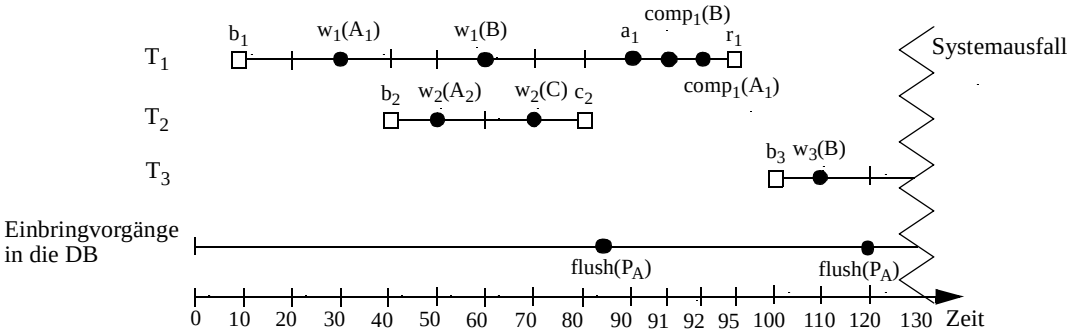
c) Führen Sie mit Hilfe der Log-Datei aus b) die in der Vorlesung vorgestellte Restart-Prozedur unter Berücksichtigung der Idempotenz der REDO-/UNDO-Operationen durch.

d) Zeichnen Sie die in der Vorlesung benutzte schematische Zustandsdarstellung der Log-Datei nach dem erfolgreichen Restart.

Lösung:

a) Man setzt CLRs für die einzelnen Änderungsoperationen der zurückgesetzten Transaktion ein, die nach der Abort-Operation (a_i) rückwärts ausgeführt werden.
Die CLRs, die Undo-Operationen entsprechen, werden ebenso als Log-Sätze protokolliert.
Bei jeder Undo-Operation wird die CLR-LSN der Seite zugewiesen, damit erkannt wird, dass beim Restart keine Änderungen zu wiederholen sind.
Bei Recovery werden Undo-Operationen in der Undo-Phase **nur** bei Transaktionen ausgeführt, die zum Systemabsturzzeitpunkt noch aktiv waren, für die also weder ein Commit- noch ein Rollback-Log-Satz vorgefunden wurde.
Bemerkung: Die Seiten-LSN wird nicht auf die alte LSN zurückgesetzt, da sonst nachfolgende Änderungen von anderen TAs in derselben Seite verloren gehen.

b) Durchführung der Aktionen:



Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
10	b_1			1, T_1, BOT, 0, 0	
30	$w_1(A_1)$	P_A, 2		2, T_1, U/R(A_1), 1, 0	
40	b_2			3, T_2, BOT, 0, 0	
50	$w_2(A_2)$	P_A, 4		4, T_2, U/R(A_2), 3, 0	
60	$w_1(B)$	P_B, 5		5, T_1, U/R(B), 2, 0	
70	$w_2(C)$	P_C, 6		6, T_2, U/R(C), 4, 0	
80	c_2			7, T_2, EOT, 6, 0	1, 2, 3, 4, 5, 6, 7
85	flush(P_A)		P_A, 4		
90	a_1				
91	$comp_1(B)$	P_B, 8		8, T_1, -/U(B), 5, 2	
92	$comp_1(A_1)$	P_A, 9		9, T_1, -/U(A_1), 8, 1	
95	r_1			10, T_1, Rollback, 9, 0	
100	b_3			11, T_3, BOT, 0, 0	
110	$w_3(B)$	P_B, 12		12, T_3, U/R(B), 11, 0	
120	flush(P_A)		P_A, 9		8, 9, 10, 11, 12

c) Mit Hilfe der Log-Datei aus (b) führen Sie die in der Vorlesung vorgestellte Restart-Prozedur unter Berücksichtigung der Idempotenz der Undo-Operationen.

(1) Analyse-Phase

(i) Gewinner: T_1 und T_2 (alle TAen, die zum Zeitpunkt des Systemabsturzes nicht mehr aktiv waren)

(ii) Verlierer: T_3

(iii) Relevante Seiten: P_A, P_B, und P_C

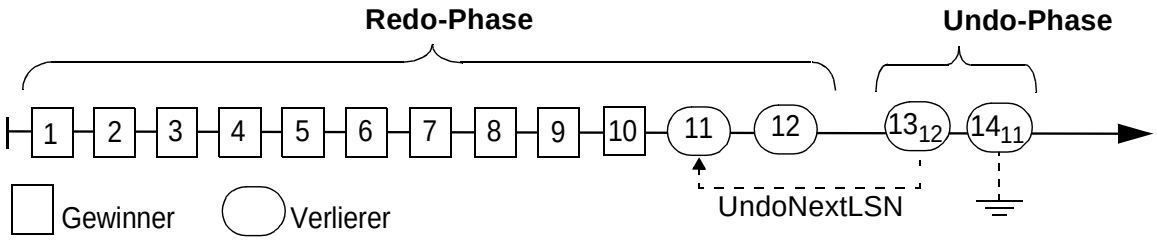
(2) Vollständige Redo-Phase: Prüfe alle Log-Sätze von T_1, T_2 und T_3 inkl. CLRs vorwärts

Log-Satz-LSN	TA	Seite	Seiten- vs. Log-Satz-LSN	Aktion
2	T_1	P_A	9 !< 2	Kein Redo
4	T_2	P_A	9 !< 4	Kein Redo
5	T_1	P_B	0 < 5	Redo $w_1(B)$, LSN(P_B)=5
6	T_2	P_C	0 < 6	Redo $w_2(C)$, LSN(P_C)=6
8	T_1	P_B	5 < 8	Redo $comp_1(B)$, LSN(P_B)=8
9	T_1	P_A	9 !< 9	Kein Redo
12	T_3	P_B	8 < 12	Redo $w_3(B)$, LSN(P_B)=12

(3) Undo-Phase:
Prüfe Log-Sätze der Verlierer-Transaktionen, also TAen, die zum Zeitpunkt des Systemabsturzes noch aktiv waren, also T_3, (inkl. CLRs) rückwärts.
Für jeden Log-Satz wird die zugehörige Undo-Operation ausgeführt und dann entsprechend ein CLR in der Log-Datei angelegt.
Bei der Ausführung eines CLR springt man zum UndoNxtLSN und setzt die Undo-Operation fort.
Hier enthält die Redo-Info die Undo-Aktion.

TA	Log-Satz-LSN	Aktion
T_3	12	Undo $w_3(B)$ und lege CLR [13, T_3, -/U(B), 12, 11] an.
T_3	11	Undo b_3 und lege CLR [14, T_3, Rollback, 13, 0] an.

d) Schematische Zustanddarstellung der Log-Datei nach erfolgreichem Restart:



Aufgabe 4: Logging und Recovery mit Sicherungspunkten

In der folgenden Tabelle sei eine Historie gegeben, in der ein Sicherungspunkt (Checkpoint *cp*) erzeugt wurde. Das DBMS stürzt nach der Ausführung der letzten Operation ab und setzt vollständiges Redo bei der Recovery ein.

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1				
z_2	$w_1(A)$				
z_3	$w_1(B)$				
z_4	c_1				
z_5	flush(P_A)				
z_6	b_2				
z_7	$w_2(A)$				
z_8	$w_2(C)$				
z_9	cp				
z_{10}	c_2				
z_{11}	b_3				
z_{12}	flush(P_A)				
z_{13}	$w_3(A)$				
z_{14}	$w_3(B)$				
z_{15}	flush(P_B)				
z_{16}	$w_3(C)$				

Vervollständigen Sie die angegebene Tabelle und führen Sie anschließend den Restart durch, wenn jeweils folgende Sicherungspunktschemata angewendet werden:

a) aktionskonsistente Sicherungspunkte (ACC)

b) Fuzzy Checkpoints

Lösung:

a) ACC:

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1			1, T_1, BOT, 0, 0	
z_2	$w_1(A)$	P_A, 2		2, T_1, U/R(A), 1, 0	
z_3	$w_1(B)$	P_B, 3		3, T_1, U/R(B), 2, 0	
z_4	c_1			4, T_1, EOT, 3, 0	1, 2, 3, 4
z_5	flush(P_A)		P_A, 2		
z_6	b_2			5, T_2, BOT, 0, 0	
z_7	$w_2(A)$	P_A, 6		6, T_2, U/R(A), 5, 0	
z_8	$w_2(C)$	P_C, 7		7, T_2, U/R(C), 6, 0	
z_9	cp		(P_A, 6), (P_B, 3), (P_C, 7)	8, -, CP(T_2), 0, 0	5, 6, 7 8
z_{10}	c_2			9, T_2, EOT, 7, 0	9
z_{11}	b_3			10, T_3, BOT, 0, 0	
z_{12}	flush(P_A)				
z_{13}	$w_3(A)$	P_A, 11		11, T_3, U/R(A), 10, 0	
z_{14}	$w_3(B)$	P_B, 12		12, T_3, U/R(B), 11, 0	
z_{15}	flush(P_B)		P_B, 12		10, 11, 12
z_{16}	$w_3(C)$	P_C, 13		13, T_3, U/R(C), 12, 0	

Restart bei vollständigem Redo:

(1) Analyse-Phase:

(i) Gewinner: T_2 (alle TAen, die seit letztem Checkpoint aktiv waren und zum Zeitpunkt des Systemabsturzes nicht mehr aktiv waren)

(ii) Verlierer: T_3

(iii) Relevante Seiten: P_A, P_B, und P_C

(2) Vollständige Redo-Phase:
Prüfe Log-Sätze alle TAen ab dem letzten Sicherungspunkt vorwärts.

Log-Satz-LSN	TA	Seite	Seiten- vs. Log-Satz-LSN	Aktion
11	T_3	P_A	6 < 11	Redo $w_3(A)$, LSN(P_A)=11
12	T_3	P_B	12 !< 12	Kein Redo

(3) Undo-Phase:
Prüfe Log-Sätze der Verlierer-Transaktionen, also TAen, die zum Zeitpunkt des Systemabsturzes noch aktiv waren, also T_3, rückwärts.
Für jeden Log-Satz wird die zugehörige Undo-Operation ausgeführt und dann entsprechend ein CLR in der Log-Datei angelegt.
Bei der Ausführung eines CLR springt man zum UndoNxtLSN und setzt die Undo-Operation fort.
Hier enthält die Redo-Info die Undo-Aktion.

TA	Log-Satz-LSN	Aktion
T_3	12	Undo $w_3(B)$ und lege CLR [13, T_3, -/U(B), 12, 11] an.
T_3	11	Undo $w_3(A)$ und lege CLR [14, T_3, -/U(A), 13, 10] an.
T_3	10	Lege CLR [15, T_3, Rollback, 14, 0] an.

b) Fuzzy Checkpoints:

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1			1, T_1, BOT, 0, 0	
z_2	$w_1(A)$	P_A, 2		2, T_1, U/R(A), 1, 0	
z_3	$w_1(B)$	P_B, 3		3, T_1, U/R(B), 2, 0	
z_4	c_1			4, T_1, EOT, 3, 0	1, 2, 3, 4
z_5	flush(P_A)		P_A, 2		
z_6	b_2			5, T_2, BOT, 0, 0	
z_7	$w_2(A)$	P_A, 6		6, T_2, U/R(A), 5, 0	
z_8	$w_2(C)$	P_C, 7		7, T_2, U/R(C), 6, 0	
z_9	cp			8, -, CP((P_A,6), (P_B,3), (P_C,7), {T_2}MinD-PLSN=3), 0, 0	5, 6, 7, 8
z_{10}	c_2			9, T_2, EOT, 7, 0	9
z_{11}	b_3			10, T_3, BOT, 0, 0	
z_{12}	flush(P_A)		P_A, 6		10
z_{13}	$w_3(A)$	P_A, 11		11, T_3, U/R(A), 10, 0	
z_{14}	$w_3(B)$	P_B, 12		12, T_3, U/R(B), 11, 0	
z_{15}	flush(P_B)		P_B, 12		11, 12
z_{16}	$w_3(C)$	P_C, 13		13, T_3, U/R(C), 12, 0	

c) Restart bei vollständigem Redo:

(1) Analyse-Phase:

(i) Gewinner: T_1 und T_2 (alle TAs, die zum Zeitpunkt des Systemabsturzes nicht mehr aktiv waren)

(ii) Verlierer: T_3

(iii) Relevante Seiten: P_A, P_B, und P_C

(2) Vollständige Redo-Phase:
Prüfe Log-Sätze aller TAen (T_1, T_2 und T_3) ab MINDirtyPageLSN vorwärts. MINDirtyPageLSN ist die R2-Recovery-Grenze und wird im Sicherungspunkt ermittelt. Sie bezeichnet die LSN der ältesten Änderung einer seither nicht mehr ausgeschriebenen Seite. (Sie hat also nichts mit der PageLSN zu tun!)..

Log-Satz-LSN	TA	Seite	Seiten- vs. Log-Satz-LSN	Aktion
3	T_1	P_B	12 !< 3	Kein Redo
6	T_2	P_A	6 !< 6	Kein Redo
7	T_2	P_C	0 < 7	Redo $w_2(C)$, $LSN(P_C)=7$
11	T_3	P_A	6 < 11	Redo $w_3(A)$, $LSN(P_A)=11$
12	T_3	P_B	12 !< 12	Kein Redo

(3) Undo-Phase:
Wie Teil (a)