Abhängigkeiten von Systemkomponenten in Datenbanksystemen

Theo Härder, Andreas Reuter
FB Informatik
TH Darmstadt

1. Einleitung

Die Datenbankliteratur ist reich an Vorschlägen zur optimalen Implementierung einzelner Komponenten eines Datenbanksystems. Bei der Analyse einschlägiger Literaturübersichten, wie z.B. [Sch78] , finden sich zahlreiche Arbeiten, in denen Algorithmen für folgende Aufgaben vorgestellt bzw. auf ihre Leistungsfähigkeit untersucht werden:

- Optimierung von Zugriffspfadstrukturen
- Verwaltungsstrategien für Externspeicher, Clusterungstechniken usw.
- Bestimmung optimaler Sicherungspunkt-Intervalle
  Hier werden meist stark vereinfachte Warteschlangenmodelle auf der Grundlage idealisierter Annahmen betrachtet; Analysen implementierter Systeme sind rar.
- Sperrstrategien, Deadlockbehandlung und Konsistenzsicherungsmaßnahmen
- Verwaltung des Datenbankpuffers

Die meisten dieser Arbeiten behandeln ihr Thema so, als könne es losgelöst von den anderen Bestandteilen eines Datenbanksystems betrachtet werden - wobei diese Voraussetzung niemals explizit genannt wird. Die Entscheidung allerdings, ob ein für sich gesehen vielversprechendes Verfahren sich zur Verwendung in einer wesentlich komplexeren Struktur eignet, setzt die genaue Kenntnis aller seiner Nebenwirkungen und Abhängigkeiten voraus.

Systematische Untersuchungen hierzu sind bis jetzt sehr selten [BS77]; Hinweise auf die Art und Auswirkung solcher wechselseitigen Abhängigkeiten finden sich eher implizit in Darstellungen über Eigenschaften und Leistungsaspekte existierender Systeme [Gr79],[St76].

Diese Arbeit unternimmt den Versuch, einige der wichtigsten Wechselwirkungen zwischen den Funktionsgruppen eines Datenbanksystems (DBS) zu benennen und ihre Konsequenzen kurz aufzuzählen. Wir werden dabei zwei Arten von Beziehungen unterscheiden:

a) die konzeptionellen Abhängigkeiten im zentralen Bereich eines DBS; das sind zwingende Konsequenzen, die sich aus Entwurfsentscheidungen zur Implementierung einer Komponente auf die Implementierung der anderen ergeben,

b) Abhängigkeiten unter Leistungsaspekten zwischen den zentralen Funktionen und den übrigen Komponenten des DBS einerseits und dem Betriebssystem andererseits; damit sind Abstimmungen zwischen Komponenten gemeint, die zum Funktionieren des Systems nicht notwendig, zur Optimierung seiner Leistung aber wünschenswert sind.

All diese Aspekte können in einem so beschränkten Rahmen natürlich eher aufgezählt denn diskutiert werden.

2. Wechselwirkungen zwischen den zentralen DBS-Komponenten

Die in Abb. 1 aufgeführten Komponenten eines DBS werden als zentral in dem Sinne aufgefaßt, daß sie alle Betriebsmittel eines Datenbanksystems auf der untersten Abstraktionsebene, d.h. unmittelbar an der Schnittstelle zum Betriebssystem, verwalten. Alle Datentypen auf höheren Stufen der Abstraktion, von Zugriffspfaden über logische Sperren bis hin zu benutzerdefinierten rücksetzbaren Transaktionen benutzen zu ihrer Realisierung die von diesen vier Komponenten bereitgestellten elementaren Datenobjekte und Operationen.

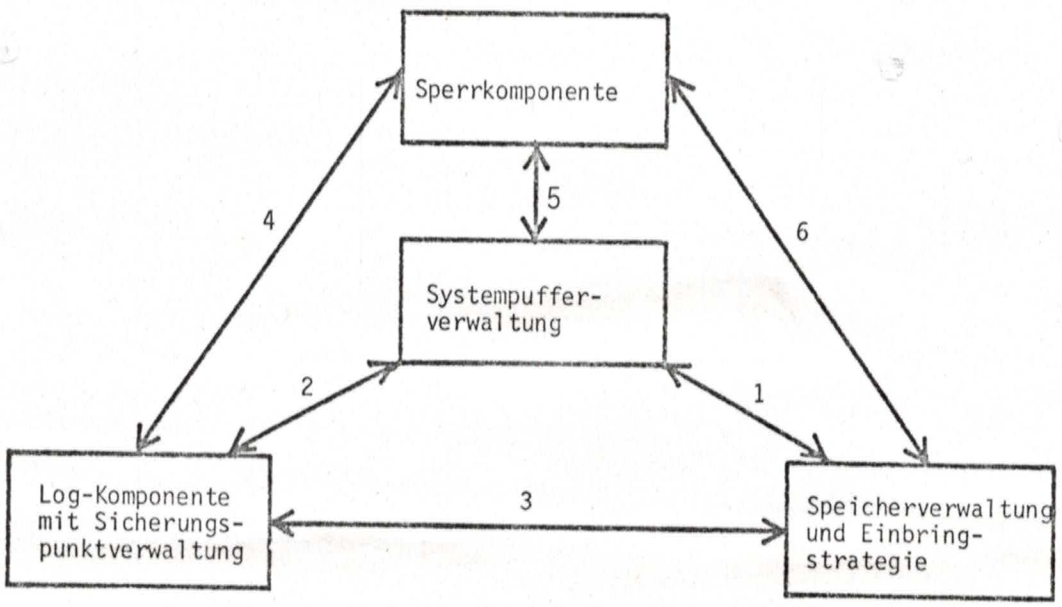

Abb.1: Konzeptionelle Abhängigkeiten zwischen den zentralen Komponenten eines DBMS

Natürlich ist es im Rahmen dieses Papiers nicht möglich, Funktionsweise und -umfang jeder der Komponenten ausführlich zu definieren; wir werden vielmehr eine stichwortartige Kurzbeschreibung verbunden mit Hinweisen auf die einschlägige Literatur geben und uns ansonsten auf ein mehr oder weniger detailliertes Vorverständnis des Lesers stützen.

- *Sperrkomponente:* Die Sperrkomponente hat im Mehrbenutzerbetrieb für alle parallel ablaufenden Transaktionen n.M. den logischen Einbenutzerbetrieb zu gewährleisten, d.h. jede Transaktion soll so arbeiten können, als sei sie alleiniger Benutzer der Datenbank; s.[EGLT76],[GLPT76]. Die wichtigste Bedingung hierfür ist die Einhaltung eines zweiphasigen Sperrprotokolls, das eine Wachstumsphase umfaßt, in der nur Sperren angefordert werden dürfen, und eine Schrumpfungsphase, in der nur erworbene Sperren freigegeben werden dürfen. Die Unterscheidung logischer Sperren, bei denen die Menge der zu sperrenden Datenobjekte durch ein Prädikat beschrieben wird, und physischer Sperren, bei denen jedes einzelne Datenobjekt und mit ihm alle darin enthaltenen (Datei, Satztyp, Satz, Feld) explizit gesperrt werden, ist für unsere Diskussion nicht von Belang, da jede bekannte Implementierung logischer Sperren auf der Benutzerschnittstelle auf einer tieferen Ebene physische Sperren benutzt [Hä78] . Die kleinsten mit einer Sperre zu belegenden Einheiten (Granulate) sind in den existierenden DBS verschieden; üblich sind Sperren in Einheiten von logischen Sätzen (Tupel-Sperren) oder physischen Blöcken (Seitensperren), Vor- und Nachteile beider Methoden werden in [RS79] diskutiert.

- *Systempufferverwaltung:* Im Systempuffer eines DBS werden die zum Lesen oder Ändern benötigten Seiten der Datenbank im Hauptspeicher bereitgestellt. Die in der "Benutzt"-Hierarchie oberhalb der Pufferverwaltung stehenden Komponenten fordern bei dieser eine bestimmte Seite an (logische Referenz, s. [Ef79]). Wenn die Seite nicht schon im Puffer steht, muß die Pufferverwaltung, sofern kein freier Platz mehr vorhanden ist, eine Seite auswählen, die ersetzt werden soll; hierfür werden meist "least recently used"-(LRU) Algorithmen verwendet. Wurde diese Seite geändert, muß bei der Externspeicherverwaltung ihr Rückschreiben in die Datenbank veranlaßt und danach die angeforderte Seite eingelesen werden.Die Probleme der Seitenersetzung werden in [RR76] und [SB76] diskutiert.

- *Log-Komponente und Sicherungspunkt-Verwaltung:* Beide Komponenten haben die Aufgabe, Informationen zu sammeln und an einem sicheren Platz zu speichern, die zum Rücksetzen einer nicht erfolgreichen Transaktion, zur Wiederherstellung eines konsistenten Zustandes nach einem Systemzusammenbruch und zum Nachholen aller vollständigen Transaktionen nach dem Verlust einer Magnetplatte erforderlich sind [Gr78]. Hierfür können die DML-Befehle samt ihrer Argumente benutzt werden (logisches Übergangs-Logging) oder es werden die Inhalte geänderter Datenobjekte vor bzw. nach der Änderung aufgezeichnet (physisches Zustands-Logging). Ein Sicherungspunkt (SP) dient dazu, den Aufwand zur Wiederholung vollständiger Transaktionen nach einem Systemzusammenbruch zu begrenzen. Man unterscheidet, abhängig vom Zeitpunkt ihrer Erzeugung, drei Arten von Sicherungspunkten:
speicherkonsistente SP's, wenn quer durch laufende Transaktionen ein SP dann geschrieben wird, wenn kein ändernder DML-Befehl aktiv ist; logisch konsistente SP's, die erzeugt werden, wenn keine Änderungstransaktion aktiv ist; transaktions-

bezogene SP's, die am Ende einer Änderungstransaktion deren Modifikationen sicherstellen. Näheres hierzu findet sich in [HR79b].

- *Speicherverwaltung:* Hierüber werden die von der Pufferverwaltung ausgelösten physischen Lese- und Schreiboperationen abgewickelt. Von besonderem Interesse ist in unserem Zusammenhang die zum Zurückschreiben geänderter Seiten angewendete Strategie. Es gibt das direkte Einbringen von Änderungen, meist als "update in place" bezeichnet (IMS, UDS), sowie das verzögerte Einbringen zum Sicherungspunkt mittels sog. "side file"-Konzepte ([SL76],[Lo77])für dateibezogene Sicherung oder aber transaktionsbezogene wie in [HR79a] und [Re79]. Die Konsequenzen verschiedener Einbringstrategien werden in [Gr79] und [HR79b] ausführlich erörtert.

Die Verbindungslinien in Abb. 1 sind nicht als Pfade im Sinne einer "Benutzt"-Hierarchie zu verstehen. Vielmehr bezeichnen sie konzeptionelle Abhängigkeiten zwischen den zur Implementierung der einzelnen Komponenten in Frage kommenden Algorithmen, deren Untersuchung und Bewertung eigentlicher Gegenstand dieser Arbeit ist.

2.1 Wechselwirkungen zwischen Systempuffer- und Externspeicherverwaltung

Betrachtet man die Seitenersetzung als eigentliche Aufgabe der Pufferverwaltung, dann sind vor allem die Einflüsse anderer Komponenten auf die hierfür üblicherweise verwendeten Strategien interessant. Ein vollkommen isolierter Ersetzungsalgorithmus z.B. nach LRU würde jede logische Referenz auf eine (vorhandene) Seite gleich behandeln und die betreffende Seite an die Spitze der LRU-Kette setzen.
Bedenkt man nun aber das isolierte Zurücksetzen einer Transaktion unter Benutzung physischer Before-Images, dann ist leicht einzusehen, daß die daraus sich ergebenden Referenzen vom Pufferverwalter eigentlich anders behandelt werden sollten:
Da jede geänderte Seite (in umgekehrter Änderungsreihenfolge) von der Protokolldatei ersetzt wird, hat jede einzelne eine geringe Wiederbenutzungswahrscheinlichkeit und sollte nicht an die Spitze der LRU-Kette. Einige Pufferersetzungsstrategien unterscheiden geänderte und nur gelesene Seiten, da die Ersetzung einer geänderten Seite wegen des damit verbundenen Schreibens aufwendiger ist als das einer nur von Lesern benutzten. Dafür wird in bestimmten Abständen der ganze Puffer ausgeschrieben (d.h. alle geänderten Pufferseiten). Ein Beispiel hierfür ist das DBS ADABAS [ADAB]. Auch hier muß eine Verbindung zwischen Puffer- und Speicherverwaltung bestehen.

Die stärkste Wechselwirkung erfordern Einbringstrategien, die einen transaktionsbezogenen Sicherungspunkt ermöglichen. Hierbei werden die durch dieselbe Transaktion geänderten Seiten durch Vorwärtszeiger miteinander verkettet, was bedeutet, daß von jeder Änderungstransaktion zu jedem Zeitpunkt (außer der Ende-Be-

handlung) mindestens eine Seite im Puffer bleiben muß. Diese Notwendigkeit durchbricht jede rein an den logischen Seitenreferenzen orientierte Ersetzungsstrategie. Auch dieser Aspekt wird in der oben zitierten Literatur diskutiert.

2.2 Wechselwirkungen zwischen Systempuffer- sowie Log- und Sicherungspunkt-Verwaltung

Die wohl wichtigste Verknüpfung zwischen beiden Komponenten wird durch das sog. "write ahead log"-(WAL) Prinzip festgelegt (s.[Gr78]). Dieses Prinzip besagt in der in[HR79b] verallgemeinerten Version, daß Log-Informationen, die dem Rücksetzen unvollständiger Transaktionen dienen, geschrieben werden müssen, bevor die entsprechenden Änderungen in die Datenbank eingebracht werden. Bei dem gängigen update in place mit physischem Zustands-Logging bedeutet dies, daß der alte Wert (before image) auf die Protokolldatei geschrieben werden muß, bevor die Seite selbst geschrieben werden kann. Da diese Forderung u.U. zu einer Engpaß-Situation an der Log-Datei führen kann, die durch Pufferung einzelner Log-Einträge gemildert werden könnte, bedeutet eine derartige Optimierung eine "Störung" jeder isolierten Puffer-Ersetzungsstrategie.

Ähnliches gilt bei der Erzeugung segmentorientierter Sicherungspunkte in Verbindung mit indirekten Einbringstrategien. Hier müssen zum Sicherungspunkt alle geänderten Seiten ausgeschrieben werden (was ein Schreiben aller Rücksetzinformationen erzwingt), wodurch ebenfalls - wie dies schon in 2.1 in Zusammenhang mit der Transaktions-Ende-Behandlung erläutert wurde - Einfluß auf die Pufferverwaltung genommen werden kann.

2.3 Wechselwirkungen zwischen Log-Komponente und Externspeicherverwaltung

Die Einbringstrategie der Speicherverwaltung beeinflußt maßgeblich die Anzahl der für die Log-Komponente anwendbaren Verfahren. Grundsätzlich gilt, daß bei allen Methoden des direkten Einbringens von Änderungen logisches Logging zum Zurücksetzen unvollständiger Transaktionen nach einem Systemzusammenbruch nicht möglich ist, da die Ausführung (in diesem Falle inverser) DML-Befehle einen speicherkonsistenten Zustand der Datenbank voraussetzt, der bei update in place nach einem abrupten Anhalten des Systems im allgemeinen nicht gegeben ist. Bei indirekten Einbring-Methoden können alle Log-Verfahren benutzt werden (soweit sie überhaupt erforderlich sind; s.[HR79b]).

Weiterhin hat die Einbringstrategie eine Rückwirkung auf die Log-Komponente, die sich weiter bis auf den Systempuffer erstreckt (s.2.2). Bei direktem Einbringen der Änderungen gilt das strikte WAL-Prinzip, bei indirektem Einbringen können geänderte Seiten beliebig zurückgeschrieben werden, das "write ahead" bezieht sich hier ledig-

lich auf den jeweils nächsten Sicherungspunkt.

Schließlich gibt es noch einen Zusammenhang zwischen der physischen Blockgröße und dem Log-Granulat bei physischem Logging (d.i. die Größe der protokollierten Zustandsinformationen): Wenn bei direktem Einbringen das Log-Granulat kleiner ist als die physische Blockgröße, kann schon ein auf einen Block beschränkter Schreibfehler den Verlust der Datenbank mit Wiederholen aller Transaktionen ausgehend von der letzten Kopie bedeuten.

2.4 Wechselwirkungen zwischen Log- und Sperrkomponente

Das Grundgesetz, das die Beziehungen beider regelt, lautet: Das Sperrgranulat muß mindestens so groß sein wie das Log-Granulat. Im Falle von logischem Logging ist das Log-Granulat die Menge aller durch den DML-Befehl geänderten Datenobjekte. Das wird unmittelbar verständlich, wenn man bedenkt (in diesem Beispiel sei physisches Logging unterstellt), daß beim Rücksetzen einer Transaktion mit Hilfe der before images die alten Werte der Datenobjekte restauriert werden, wobei alle Werte verändert werden, die auf dem Protokoll stehen; somit müssen - und zwar nur wegen der Möglichkeit des Rücksetzens - auch alle diese Werte gesperrt werden. Das bedeutet insbesondere, daß, wenn ganze Seiten auf die Log-Datei geschrieben werden, auch ganze Seiten gesperrt werden müssen.

Wenn das DBS die Möglichkeit transaktions-interner Sicherungspunkte bietet, dann müssen zu diesen Sicherungspunkten alle in diesem Moment gehaltenen Sperren zusätzlich zu den Zustandsinformationen der Datenobjekte protokolliert werden; näheres hierzu siehe z.B. in [Gr78].

Wenn die Sperrkomponente nur ein einfaches 2-Phasen-Protokoll befolgt anstelle des strikten 2-Phasen-Protokolls (Freigabe aller Sperren erst am Ende der Transaktion), müssen alle Interferenzen zwischen Transaktionen (auch solchen, die nur lesen) über gemeinsam genutzte Datenobjekte protokolliert werden, um ein rekursives Zurücksetzen im Fehlerfall zu ermöglichen.

2.5 Wechselwirkungen zwischen Sperrkomponente und Systempuffer-Verwaltung

Neben der zentralen Aufgabe, den logischen Einbenutzerbetrieb für alle Transaktionen sicherzustellen, muß die Sperrkomponente in Zusammenarbeit mit der Pufferverwaltung weiter gewährleisten, daß die nach der Anforderung einer Seite durch irgendein Modul an diese übergebene Pufferadresse der Seite solange gültig bleibt (Fix-Phase), bis der Modul ausdrücklich auf die weitere Benutzung der Seite verzichtet (Unfix). Da eine Seite während der Fix-Phase der Verfügung der Puffer-Verwaltung teilweise

entzogen ist, (sie kann zwar ihre Position in einer LRU-Kette ändern, darf aber nicht ersetzt werden), kann es insbesondere bei langen Fix-Phasen zu Betriebsmittel-Mangel im Puffer kommen, wenn ein Modul auf diese Weise alle Seiten im Puffer festhält und eine weitere Seite anfordert, die nicht im Puffer steht, die aber auch nicht bereitgestellt werden kann, da eben wegen dieses Moduls keine der vorhandenen Seiten ersetzt werden darf. Diese Wechselwirkung hat weitreichende Konsequenzen für die Modulstruktur des gesamten DBS. Das Bestreben, solche Situationen zu vermeiden und alle Fix-Phasen möglichst kurz zu halten, kann andererseits zu einer "Übermodularisierung" mit erheblichen Leistungsverlusten führen.

2.6 Wechselwirkungen zwischen Sperrkomponente und Speicherverwaltung

In existierenden Datenbanksystemen sind beide Komponenten im Prinzip unabhängig; die hier zu diskutierende Beziehung ergibt sich aus einigen bislang nur theoretisch untersuchten Implementierungen . In [Ba79] wird ein Sperrkonzept mit dem Ziel erhöhter Parallelität vorgeschlagen, was darauf beruht, daß Datenobjekte, die von Änderungstransaktionen gesperrt sind, gleichzeitig ablaufenden Lesetransaktionen in Gestalt ihres jew. before image zugänglich gemacht werden können. Das setzt voraus, daß die Speicherverwaltung eine Art transaktionsorientiertes Schattenspeicherkonzept unterstützt, wie es etwa in [HR79a] und [Re79] beschrieben wird. Da diese Verfahren auf Ketten von Schattenseiten beruhen, die jeweils allen von einer Transaktion geänderten Seiten entsprechen, müssen, aus dem gleichen Grund, der in 2.3 die Beziehung zwischen Log- und Sperr-Granulaten festgelegte, bei den genannten Algorithmen Seiten als kleinste Einheit gesperrt werden.

3. Abhängigkeiten höherer Systemfunktionen

Die bisher diskutierten Wechselwirkungen zwischen den zentralen Komponenten eines DBS sind Abhängigkeiten zwischen Konzepten und deshalb beim Systementwurf in jedem Fall zu berücksichtigen, um die gewünschten Funktionen gewährleisten zu können.
Es sollen jetzt Abhängigkeiten zwischen Systemkomponenten betrachtet werden, die vor allem die Performance des Systems beeinflussen. Diese Abhängigkeiten sind nicht konzeptbedingt, d.h. die Abstimmung des Zusammenwirkens dieser Komponenten ist nicht prinzipiell erforderlich; aus Gründen der Wirksamkeit kann jedoch darauf im praktischen Einzelfall nicht verzichtet werden. Für die beispielhafte Betrachtung solcher optimierungsbedingter Abhängigkeiten beschränken wir uns auf die in Abb. 2 dargestellten Beziehungen zwischen Systemkomponenten.

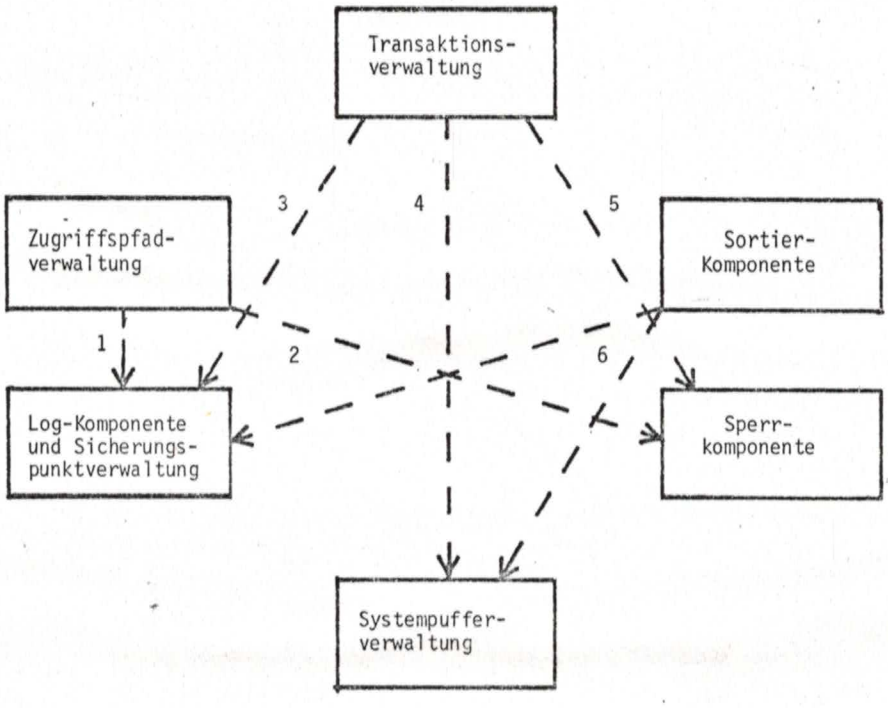

Abb. 2: Optimierungsbedingte Abhängigkeiten höherer Systemkomponenten

- *Zugriffspfadverwaltung:* Die Zugriffspfadverwaltung hat eine Reihe von Strukturen bereitzustellen, die das Aufsuchen von Datensätzen in wirksamer Weise unterstützen. Dazu zählen Zugriffspfade für Primärschlüssel und Sekundärschlüssel, sowie für Beziehungen hierarchischer Art zur Verknüpfung von Datensätzen verschiedenen Typs (Setstrukturen nach CODSAYL). In einer Datenbank sind gewöhnlich eine Vielzahl solcher Zugriffspfade verschiedenen Typs zu warten, die bei Änderungsoperationen auf Datensätzen automatisch durch die Zugriffspfadverwaltung zu aktualisieren sind. Einen Überblick über häufig in DBS eingesetzte Zugriffspfadstrukturen findet man in [WH76] und [Sch78] . Von besonderer Wichtigkeit für DB-Anwendungen sind Indexstrukturen, die den schnellen direkten Zugriff über Schlüssel und den sortiert sequentiellen Zugriff zu allen Datensätzen eines Typs erlauben. Als Implementierungstechniken eignen sich vor allem B^*-Bäume wegen ihrer dynamischen Erweiterbarkeit durch Splittechniken.

- *Transaktionsverwaltung:* Durch das DBS ist pro Benutzer eine Transaktion zu verwalten. Sie dient dem DBS als Einheit der Zuteilung DBS-interner Betriebsmittel, des Scheduling von Anforderungen, der physischen Wiederherstellung von Daten im Fehlerfall und der Erhaltung der logischen Konsistenz der Datenbank. Eine Transaktion, bestehend aus einer beliebig langen Folge von DML-Operationen eines Benutzers, führt bei erfolgreichem Abschluß die Datenbank von einem gültigen in einen neuen gültigen Zustand über, wobei idealerweise das DBS die Zulässigkeit des Zustandsüberganges mit Hilfe von sogenannten Zusicherungen oder Konsistenzbedingungen [EC75] kontrolliert. Änderungen von Transaktionen, die aus Gründen von Konsistenzverletzungen oder durch Eintreten von Fehlerzuständen nicht erfolgreich abgeschlossen werden, müssen so zurückgesetzt werden, als wären die Transaktionen nie ge-

startet worden. Die Transaktionsverwaltung hat den parallelen Ablauf von Transaktionen zu kontrollieren, bei erfolgreichem Abschluß (EOT) die Betriebsmittel freizugeben und im Fehlerfall geeignete Recovery-Aktionen anzustoßen.

- *Sortierkomponente:* Datenbanksysteme, die eine Spezifikation von Benutzeranforderungen in einer deskriptiven Sprache erlauben, müssen durch spezielle Algorithmen den Zugriff auf Mengen von Datensätzen optimieren. Um solche Anforderungen effizient abwickeln zu können, ist es erforderlich, intern Mengen von Datensätzen umzuordnen, beispielsweise zur Unterstützung komplexer Relationsoperationen [Hä78]. Aus Gründen der Benutzerfreundlichkeit sollte außerdem die Spezifikation einer Ausgabefolge der qualifizierten Datensätze angeboten werden. Beide Arten von Benutzeranforderungen setzen eine wirksame Sortierkomponente des DBS voraus.

3.1 Auswirkungen der Log-Funktion auf die Zugriffspfadverwaltung

Die Art der Log-Information, die die Log-Komponente als redundante Information für den Fehlerfall auf einem sicheren Platz sammelt, hat erhebliche Auswirkungen auf das Zeitverhalten von Änderungsoperationen des DBS. Am Beispiel von Änderungen in B^*-Bäumen als redundante Zugriffspfade soll dieser Einfluß kurz diskutiert werden. Für den praktischen Einsatz sind vor allem zwei Log-Techniken relevant [HR79b]: Das logische Übergangs-Logging und das physische Zustands-Logging (s. Kap. 2).

Nehmen wir an, daß ein Datensatz, für den über n Attribute Zugriffspfade vorhanden sind, neu in die DB eingebracht werden soll. Neben dem eigentlichen Einfügen des Datensatzes müssen durch die Zugriffspfadverwaltung n Indexstrukturen - typischerweise B^*-Bäume - aktualisiert werden, damit sie auf den neuen Satz verweisen. Logisches Übergangslogging erfordert für die Wartung der Zugriffspfade keinerlei zusätzlichen Log-Aufwand, da durch die Protokollierung der Einfügeoperation - unter der Annahme einer indirekten Einbringstrategie (s. 2.3) - die Operation im Fehlerfall jederzeit zurückgesetzt oder wiederholt werden kann.

Physisches Zustandslogging dagegen erzwingt bis zu 2n Schreibvorgänge (abhängig von möglichen Blockungstechniken) für zugriffspfadbezogene Log-Informationen.

Ein Musterbeispiel für extrem hohe verborgene Kosten, die oft durch Implementierungsdetails eingeführt werden, soll die Effizienzabhängigkeit der betrachteten Komponenten verdeutlichen. Zur Unterstützung von navigierenden Operationen könnte in B^*-Bäumen für jeden Knoten ein Rückwärts-Pointer zum Vater vorgesehen werden. Beim Splitten eines Vaterknotens bringt diese "unscheinbare Maßnahme" hohe Folgekosten durch die notwendige Aktualisierung der Vater-Pointer in den Söhnen mit sich. Verstärkt wird dieser Wartungsaufwand durch die Log-Kosten vor allem beim physischen Zustandslogging auf Seitenbasis.

Unter der Annahme, daß ein Vaterknoten mit 400 Söhnen gesplittet wird und der neu eingeführte Vaterknoten auf 200 Söhne verweist, lassen sich für die Aktualisierung der Vater-Pointer folgende Zusatzkosten angeben:

- 200 Seiten (Söhne) lesen und geändert zurückschreiben
- 200 Seiten Before-Images in die Log-Datei schreiben
- 200 Seiten After-Images in die Log-Datei schreiben.

Dieses Beispiel soll in drastischer Weise zeigen, daß Optimierungsmaßnahmen für Zugriffspfade in einem DBS die Abstimmung mit der Log-Funktion, der Systempufferverwaltung und der Einbringstrategie der Speicherkomponente erfordern.

3.2 Auswirkungen der Sperr-Funktion auf die Zugriffspfadverwaltung

Im Mehrbenutzerbetrieb müssen im Hinblick auf die Optimierung von Zugriffsvorgängen zusätzliche Auswirkungen weiterer Systemkomponenten berücksichtigt werden. Besonders bei B^*-Bäumen kann ein schlecht angepasstes Sperrkonzept gravierende Folgen für die Parallelität von Transaktionen haben. Exklusive Seitensperren auf der Wurzel von B^*-Bäumen können nicht toleriert werden, da dadurch alle Pfade zu den Blättern blockiert werden. Wegen der typischerweise hohen Frequenz von Suchvorgängen in B^*-Bäumen sollte ein DBS spezielle Algorithmen für das Sperren von solchen Strukturen anbieten. Eine Reihe von verklemmungsfreien Sperralgorithmen werden in [BS77] diskutiert.

3.3 Auswirkungen der Log-Funktion auf die Transaktionsverwaltung

Das Ende einer Transaktion ist der spätest mögliche Zeitpunkt zum Ausschreiben aller REDO-Informationen einer Transaktion, die erforderlich sind, um im Fehlerfall die Änderungen aller abgeschlossenen Transaktionen in ihrer ursprünglichen Reihenfolge wiederholen zu können. Das Ausschreiben der UNDO-Information zum Zurücksetzen einer unvollständigen Transaktion wird in Abhängigkeit von der Einbringstrategie durch das WAL-Prinzip oder durch die SP-Generierung vorgeschrieben (s. 2.2 und 2.3).

Die Transaktionsverwaltung hat folglich bei der EOT-Behandlung jeder Transaktion über die Log-Komponente sicherzustellen, daß die REDO-Information und ein EOT-Quittungsvermerk vor dem Abschluß der Transaktion in der Log-Datei stehen. Werden transaktionsbezogene SP erzeugt, so gilt zusätzlich der Schreibzwang bei EOT für alle geänderten Seiten einer Transaktion, so daß sowohl DB- als auch Log-Ausgabe anfällt. Solche synchronen SP belasten die Kanäle und Speichermedien in erhöhtem Maße, so daß sie potentielle Engpässe darstellen. Abhängig von der Verfügbarkeit von Kanälen wird eine *relative* Serialisierung der Ausgabe erzwungen, die sich auf die Laufzeiten

der Transaktionen auswirkt. Um solche graduellen Verstopfungen bei der EOT-Behandlung zu vermeiden, sollten ggf. asynchrone SP-Verfahren gewählt werden.

Bei hohem Verkehrsaufkommen mit kurzen Transaktionen, wie sie typischerweise bei DB/DC-Anwendungen anfallen, wird die reine Log-Ausgabe bei EOT der Systemengpaß. Bei transaktionsbezogener Ausgabe der REDO-Information auf die Log-Datei bestimmt die Geschwindigkeit des externen Speichermediums die maximale Transaktionsrate. Abhilfe in dieser Situation könnte dadurch geschaffen werden, daß die Transaktionsverwaltung die EOT-Behandlung von Transaktionen in bestimmten Zeitintervallen ($\sim$ 50-100 ms) verzögert und die Log-Komponente veranlaßt, die Log-Information aller verzögerten Transaktionen geblockt auszugeben.

3.4 Einflußmöglichkeiten der Transaktionsverwaltung auf die Systempufferverwaltung

Die Transaktionsverwaltung interpretiert die Anforderungen der Benutzer (DML-Anweisungen) und verteilt sie zur Bearbeitung an die entsprechenden Systemkomponenten. Da sich durch die Art der Operatoren und ihres Kontextes Information über eine mögliche Lokalität der Seitenreferenzen ableiten läßt, könnte durch dieses Wissen die Ersetzungsstrategie der Systempufferverwaltung beeinflußt werden. Seitenanforderungen für verschiedenartige DB-Operationen können durch charakteristische Seitenersetzungsstrategien optimiert werden [AIM] , beispielweise

- sequentielle Anforderungen durch FIFO
- zufällige Anforderungen durch LRU
- baumstrukturierte Anforderungen durch GCLOCK ("biased" LRU).

Bei vorgegebener Puffergröße hat die Anzahl paralleler Transaktionen einen erheblichen Einfluß auf die Seitenwechselrate. Dadurch, daß für jede Transaktion bei jeder DML-Operation einige Seiten im Puffer kurzfristig exklusiv gesperrt (Fix-Phase) und dabei neue Seiten zur Fortsetzung der Verarbeitung angefordert werden, kann es vor allem bei kleinen Puffergrößen zu Verklemmungssituationen (Betriebsmitteldeadlock) kommen (s. 2.5). Aber auch das gegenseitige Wegnehmen von Pufferrahmen und Verdrängen von Seiten beeinträchtigt das gesamte Systemverhalten in gravierender Weise. Über ein zeitweiliges Zurückhalten von Anforderungen durch die Transaktionsverwaltung bei drohenden Betriebsmittelengpässen läßt sich ein wirksamer Thrashing-Schutz für den Systempuffer einführen.

3.5 Abstimmung von Transaktionsverwaltung und Sperrverwaltung

Über die Sperrverwaltung wird die Zugriffssynchronisation zu den Objekten der Datenbank in zentraler Weise durchgeführt. Die Transaktionsverwaltung und ggf. andere Komponenten des DBS rufen vor dem aktuellen Zugriff entsprechend der "Benutzt"-

Hierarchie die Sperrverwaltung zum Sperren des betreffenden Objektes. Beim Rücksetzen im Fehlerfall oder bei EOT veranlaßt die Transaktionsverwaltung die Freigabe aller Sperren der betroffenen Transaktion. Wird eine Verklemmungssituation erkannt, ist eine Transaktion als "Opfer" auszuwählen und durch Zurücksetzen die zyklische Wartesituation zu brechen. Zur Minimierung des Verlustes ist eine Abstimmung zwischen Transaktionsverwaltung und Sperrverwaltung sehr hilfreich. Durch zusätzliche Informationen der Transaktionsverwaltung können so Kostenkriterien wie verbrauchte Rechenzeit und Anzahl der durchgeführten Änderungen zusammen mit der Anzahl der gehaltenen Sperren berücksichtigt werden.

3.6 Ausführung spezieller Funktionen

Im allgemeinen besitzen die Primitive der zentralen Systemkomponente nur eine geringe Tauglichkeit für die Ausführung spezieller Funktionen. Ein Musterbeispiel ist die Realisierung der Sortieroperation [Hä77], die in relationalen DBS unerläßlich ist. Es ist davon auszugehen, daß zur Unterstützung komplexer Relationenoperationen (Verbund) häufig große Mengen von Datensätzen (ganze Relationen) mit Hilfe von externen Sortier- und Mischverfahren umzuordnen sind.

Die Abwicklung solcher E/A-intensiver Operationen über die normale Systempuffer- und Log-Schnittstelle hat weitreichende Konsequenzen auf die Performance des DBS. Falls ein m-phasiges Mischen von n Datenseiten über die normale E/A-Schnittstelle durchgeführt wird, sind, explizite Auslagerung der Daten in jeder Mischphase vorausgesetzt, beim physischen Zustandslogging n.m Before-Images und n.m After-Images zu schreiben. Die LRU-Ersetzungsstrategie bei der Pufferverwaltung ist in diesem Fall auch wenig geeignet, da die Mischoperationen vorwiegend sequentieller Natur sind. Bei Einsatz eines Schattenspeicherkonzeptes durch die Speicherkomponente ergeben sich zusätzliche hohe Reibungsverluste, da dieses Konzept für das wiederholte Umspeichern ganzer Segmente, das durch jede Mischphase erzwungen wird, unverhältnismäßig aufwendig ist.

Obwohl grundsätzlich durchführbar, erfordern spezielle höhere Systemfunktionen bei einem brauchbaren Systementwurf eine detaillierte Abstimmung mit den zentralen Komponenten des DBS.

4. Schlußbemerkungen

Beim konkreten Systementwurf sind neben den Abhängigkeiten zwischen den Komponenten des DBS auch die Randbedingungen zu berücksichtigen, die durch die Systemumgebung zur Ablaufzeit vorgegeben sind. Dazu gehören

- die Beziehung zwischen DBS und Anwendungsprogramm
- die Verbindung von DBS und DC-System.

Ähnlich wie *innerhalb* des DBS sind auch *zwischen* den Systemen konzeptbedingte Wechselwirkungen und optimierungsbedingte Abhängigkeiten zu unterscheiden. Eine Reihe von Problemen, die in diesem Zusammenhang auftreten, sind in [Hä79] ausführlich untersucht worden, so daß an dieser Stelle eine Aufzählung der wichtigsten Fragen genügt.Zu den konzeptionellen Problempunkten gehören:

- Ein virtuelles E/A-Konzept des Betriebssystems (BS), bei dem E/A-Aufträge angenommen werden, die aktuelle Ausgabe aber unbestimmt ist, führt auf Unverträglichkeiten mit Log-Konzepten, die Annahmen über definierte Schreibzeitpunkte machen.
- Zugriffskontrolle durch das DBS ist nur möglich, wenn das BS eine hinreichende Isolation von DBS und Anwendungsprogramm (AP) gewährleistet.
- Die Kooperation zwischen AP und DBS setzt die Möglichkeit der Interprozeß-Kommunikation voraus.

Folgende Abhängigkeiten sind aus Performance-Gründen zu berücksichtigen:

- Die Aufteilung der Kontrolle über Betriebsmittel auf verschiedene selbständige Systeme führt hohe Verwaltungskosten ein. Ein Beispiel ist die Verwaltung des virtuellen Speichers durch das BS und des Systempuffers - im virtuellen Speicher realisiert - durch das DBS. Dadurch wird das Problem des Double Paging eingeführt; zusätzlich können sich mit unterschiedlichen Zielsetzungen optimierte Seitenersetzungsstrategien in ihrer Wirkung neutralisieren.
- Unabhängige Recovery-Mechanismen für BS und DBS können die Kosten der Vorsorgemaßnahmen für den Fehlerfall verdoppeln.
- Der vorzeitige Entzug des Rechnerkerns (preemptive scheduling) kann bei der DB-Synchronisation zu Verstopfungen führen (Konvoi-Phänomen).
- Falls DBS und DC-System als unabhängige Prozesse ablaufen, ist ein sehr hoher Aufwand an zusätzlicher Prozeßkommunikation erforderlich. Die Abstimmung bei der Recovery im Fehlerfall ist sehr umständlich.

Diese Übersicht legt nahe, daß weitere Forschungen auf diesem Gebiet unter zwei Gesichtspunkten durchgeführt werden sollten:

- Zur Unterstützung sowohl bei der Implementierung eines DBS als auch beim Entwurf einer DBS-Anwendung sind eine präzise Untersuchung der wechselseitigen Abhängigkeiten von Systemkomponenten, ihre Nachbildung in analytischen Modellen oder durch Simulation und die Voraussage ihres Verhaltens bei isolierten Optimierungsversuchen erforderlich.
- Für Zwecke der Optimierung existierender Systeme oder des Tuning konkreter Anwendungen in einer vorgegebenen Systemumgebung sind bereits beim Entwurf des DBS geeignete Meßschnittstellen vorzusehen, die im laufenden Betrieb die zur

Anwendung der oben genannten Modelle erforderlichen Parameter zu bestimmen erlauben.

Erste Ansätze zur Verknüpfung beider Aspekte werden z.B. in [SB76] und [EHR80] diskutiert.

Literaturverzeichnis:

ADAB — ADABAS Benutzer-Handbuch, Broschüre der Software AG, Darmstadt.

AIM — AIM, Advanced Information Manager, Reference Manual, Fujitsu, Japan.

Ba79 — Bayer, R.;et al.: Parallelism and Recovery in Database Systems, in: Proc. 4th Int. Conf. on Software Engineering, Munich, Sept. 1979.

BS77 — Bayer, R., Schkolnick, M.: Concurrency of Operations on B-Trees, in: Acta Informatica, Vol.9, 1977, S. 1-21.

EC75 — Eswaran, K.P., Chamberlin, D.D.: Functional Specification of a Subsystem for Database Integrity, in: Proc. 1st Int. Conf. on VLDB 1975, Framingham, Ma., Sept. 22-24, 1975, S. 48-68.

Ef79 — Effelsberg, W.: Messung und Auswertung des Seitenreferenzverhaltens von Datenbanksystemen, in: Angewandte Informatik 10/79, S. 434-440.

EGLT76 — Eswaran,K.P., Gray, J.N., Lorie R.A., Traiger, I.L.: The Notions of Consistency and Predicate Locks in a Database System, in: Comm. ACM, Vol. 19, No.11, 1976, S. 624-633.

EHR80 — Effelsberg, W., Härder, Th., Reuter, A.: Measurment and Evaluation of Techniques for Implementing COSETs - A Case Study, erscheint im Tagungsband der Int. Conference on Database Systems, Aberdeen, Juli 1980.

GLPT76 — Gray, J.N., Lorie, R.A., Putzolu, G.R., Traiger, I.L.: Granularity of Locks and Degrees of Consistency in a Large Shared Data Base, in: Modelling in Data Base Management Systems, North Holland Publ. Comp. 1976, S. 365-394.

Gr78 — Gray, J.N.: Notes on Data Base Operating Systems, in: Lecture Notes in Computer Science 60, Advanced Course on Operating Systems, Springer-Verlag, Berlin, 1978, S. 393-481.

Gr79 — Gray, J.N.; et.al.: The Recovery Manager of a Data Management System, IBM Research Report RJ2623, San Jose, Aug. 1979.

Hä77 — Härder,T.: A Scan-Driven Sort Facility for a Relational Database System, in: Proc. 3rd Int. Conf. on VLDB, Tokyo, Japan, Oct. 1977, S. 236-243.

Hä78 — Härder, T.: Implementierung von Datenbanksystemen, Carl-Hanser-Verlag, 1978.

Hä79 — Härder,T.: Die Einbettung eines Datenbanksystems in eine Betriebssystemumgebung, in: Datenbanktechnologie, J. Niedereichholz (Hrsg.), B.G. Teubner-Verlag, Stuttgart, 1979, S.9-24.

HR79a — Härder,T., Reuter, A.: Optimization of Logging and Recovery in a Database System, in: Data Base Architecture, Proc. IFIP TC2-Working Conf., Venice 1979, G. Bracchi and G.M. Nijssen (eds.), North Holland Publ. Comp., Amsterdam 1979, S. 151-168.

HR79b — Härder,T., Reuter, A.: A Systematic Framework for the Description of Transaction Oriented Logging and Recovery Schemes, DVI 79-4, Forschungsbericht TH Darmstadt, Dez. 1979.

Lo77 Lorie, R.A.: Physical Integrity in a Large Segmented Database, in: ACM TODS, Vol. 2, No.1, March 1977, S. 91-104.

Re79 Reuter, A.: Minimizing the I/O-Operations for UNDO-Logging in Database Systems, in: Proc. 5th Int. Conf. on VLDB, Rio de Janeiro, Oct. 1979, S. 164-172.

RR76 Ragaz, N., Rodriguez-Rosell, J.: Empirical Studies of Storage Management in a Data Base System, IBM Research Report, RJ1834, San Jose, July 1976.

RS79 Ries, D.R., Stonebraker, M.: Locking Granularities Revisited, in: ACM TODS, Vol. 4,No.2, 1979, S. 210-227.

SB76 Sherman, S.W., Brice, R.S.: Performance of a Database Manager in a Virtual Memory System, in: ACM TODS, Vol. 1, No. 4, 1976, S. 317-343.

Sch78 Schkolnick, M.: A Survey of Physical Database Design Methodology and Techniques, in: Proc. 4th Int. Conf. on VLDB, Berlin, Sept. 1978, S. 474-487.

SL76 Severance, D.G., Lohmann, G.M.: Differential Files: Their Application to the Maintenance of Large Databases, in: ACM TODS, Vol. 1, No.3, 1976, S. 256-267.

St76 Stonebraker, M.; et.al.: The Design and Implementation of INGRES, in: ACM TODS, Vol. 1, No.3, 1976, S. 189-222.

WH76 Wedekind, H., Härder,T.: Datenbanksysteme II, Reihe Informatik/18, Bibliographisches Institut Mannheim, 1976 .

Diese Arbeit wurde teilweise vom Bundesminister für Forschung und Technologie, Förderkennzeichen 081 5186 und von der Siemens AG unterstützt. Wir danken den Gutachtern für wertvolle Hinweise zur deutlicheren Formulierung des zentralen Anliegens dieses Papieres.