In: Handbuch der modernen Datenverarbeitung, Heft 144, Forkel-Verlag, 1988, S. 24-35.

Transaktionskonzept und Fehlertoleranz in DB/DC-Systemen

Theo Härder

Zusammenfassung

Das Transaktionskonzept in Datenbanksystemen (DBS) führte zu einem Paradigmenwechsel bei der Anwendungsprogrammierung. Seine Hauptvorteile liegen vor allem darin, daß dem Anwendungsprogramm eine fehlerfreie Sicht auf die Datenbank (DB) gewährt wird und daß es von allen Aspekten des Mehrbenutzerbetriebs isoliert wird. In diesem Aufsatz diskutieren wir die Anforderungen an die Fehlertoleranz, die mit der Übernahme des Transaktionskonzeptes in DBS und - in geringfügiger Erweiterung - in DB/DC-Systemen verbunden sind. Nach Klärung von Art und Umfang der zu behandelnden Fehler werden die hauptsächlichen Konzepte für Sicherungsvorkehrungen im Normalbetrieb und für die transaktionsorientierte Fehlerbehandlung skizziert.

1. Das Transaktionskonzept in Datenbanksystemen

Der Begriff "Transaktion" wurde Mitte der siebziger Jahre durch DBS-Forscher und -Entwickler geprägt [3]. Seit dieser Zeit hat sich die Transaktionsorientierung als wesentliches Systemmerkmal in rasanter Weise durchgesetzt. Für DBS gilt sie heute bereits als unverzichtbare Eigenschaft, ja sogar als Definitionskriterium: ein DBS ohne Transaktionskonzept verdient seine Bezeichnung nicht. Auch in anderen Systemen wie z.B. Betriebssystemen findet es zunehmend Eingang.

Was verbirgt sich nun hinter diesem Transaktionskonzept? Wir versuchen zunächst, es aus verschiedenen Blickwinkeln zu erklären. Dazu nehmen wir an, daß eine betriebliche Anwendung durch ein DBS unterstützt werden soll. Als Voraussetzung dafür sind die Objekte der zugehörigen Miniwelt und ihre Beziehungen zueinander modellhaft durch das DBS abzubilden. Dazu gehört auch die Spezifikation von Integritätsbedingungen, durch welche die "Richtigkeit" von DB-Aktualisierungen kontrolliert werden kann. Beim Nachvollzug der Vorgänge der Miniwelt durch die DB-Operationen wird dann die Einhaltung dieser Integritätsbedingungen durch das DBS gewährleistet.

1.1 Was ist Transaktionsverarbeitung?

Aus der Sicht des Anwenders läßt sich eine Transaktion dann folgendermaßen charakterisieren:

Mit einer Transaktion wird ein Vorgang der Anwendung in einem Rechensystem unterbrechungsfrei abgewickelt. Ein solcher Vorgang bildet (typischerweise) einen nicht-trivialen Arbeitsschritt (unit of work) in betrieblichen Abläufen.

Transaktionssysteme, die die skizzierte Anwendersicht unterstützen, halten mit zunehmendem Tempo Einzug in fast alle Bereiche unseres täglichen Lebens [7, 9]. Sie erwarten zur Abwicklung einer Transaktion im einfachsten Fall eine Terminal-Eingabe, wobei der Benutzer (EDV-Laie) typischerweise über eine

```
BUCHUNG:
  BEGIN-DC
  READ MESSAGE FROM TERMINAL;                                  {100 Bytes}
  EXTRACT KONTONR, DELTA, SCHALTERNR, ZWEIGSTELLENNR
  BEGIN_TRANSACTION;
    READ KONTO   KEY IS KONTONR;
    IF NOT FOUND OR KONTOSTAND + DELTA < 0 THEN
      DO
        prepare negative response;
        RESTORE_TRANSACTION
      END
    ELSE
      DO
        KONTOSTAND := KONTOSTAND + DELTA;
        REWRITE KONTO   KEY IS KONTONR;
        prepare record BUCHUNGEN;
        WRITE BUCHUNGEN;
        READ SCHALTER   KEY IS SCHALTERNR;
        KASSENSTAND := KASSENSTAND + DELTA;
        REWRITE SCHALTER   KEY IS SCHALTERNR;
        READ ZWEIGSTELLE   KEY IS ZWEIGSTELLENNR;
        GELDBESTAND := GELDBESTAND + DELTA;
        REWRITE ZWEIGSTELLE   KEY IS ZWEIGSTELLENNR;
        prepare response;
        COMMIT;
      END
    WRITE MESSAGE TO TERMINAL;                                 {200 Bytes}
  END-DC;
```

Bild 1: Transaktionsprogramm für eine Kontenbuchung

Bildschirmmaske bei seinem Dialog geführt wird. Nach einer in der Regel kurzen Berechnung erwidert das Transaktionssystem die Eingabenachricht mit einer Ausgabenachricht, die bei erfolgreichem Transaktionsverlauf gleichzeitig die Quittung des Transaktionssystems darstellt. Solche Einschritt-Transaktionen sind heute in vielen Systemen die Regel. Eine Verallgemeinerung auf Mehrschritt-Transaktionen kommt der Natürlichkeit von Arbeitsabläufen und der konsistenten Zusammenfassung von zusammengehörigen Arbeitsschritten zugute; sie stellen allerdings erhöhte Anforderungen an die Entwicklung des Transaktionssystems.

Transaktionen werden durch Programme (TAP = Transaktionsprogramm) realisiert, bei denen die gemeinsame Nutzung von Daten ein wesentliches Merkmal ist. Dieser Aspekt läßt sich wie folgt verdeutlichen:

Eine (Online-) Transaktion ist die Ausführung eines Programmes, das mit Hilfe von Zugriffen auf eine gemeinsam genutzte Datenbank eine Anwendungsfunktion (Verwaltungsaufgabe usw.) erfüllt (im Auftrag eines Online-Benutzers).

Solche TAPs werden am Arbeitsplatz zur Durchführung von häufig anfallenden, "vorausgeplanten" Routinetätigkeiten wie etwa Auskunft-, Buchungs- oder Datenerfassungsvorgängen eingesetzt, also beispielsweise zur Überweisung eines Geldbetrages, zur Platzreservierung für einen Flug oder zur Bearbeitung einer Bestellung. Zur Veranschaulichung der weiteren Betrachtungen wollen wir hier das Musterbeispiel eines TAP [1] - die Kontenbuchung - skizzieren (Bild 1).

Nach diesen allgemeinen Charakterisierungen können wir die Transaktion aus der Sicht der Datenbank (oder des DBS) definieren:

Eine Transaktion ist eine Folge von DB-Operationen (DML-Befehle), welche die Datenbank in ununterbrechbare Weise von einem logisch konsistenten Zustand in einen (neuen) logisch konsistenten Zustand überführt.

Diese Definition entspricht der ursprünglichen Begriffsbildung [3]: als genauere Bezeichnung sollte hier der Begriff "DB-Transaktion" gewählt werden. Man beachte, daß nur Zusicherungen über den DB-Zustand gemacht werden, der Zustand der Umgebung (Betriebssystem, DC-System, Anwendungen) bleibt außer acht; die Konsequenzen daraus werden insbesondere im Fehlerfall spürbar. Weiterhin umfaßt diese (technische) Definition alle Arten von DB-Interaktionen - neben kurzen Online-Transaktionen auch lange Transaktionen in Stapelprogrammen.

Das Transaktionskonzept bezieht sich also auf alle DML-Befehle (und ihre Anwirkungen auf die DB), die durch BEGIN_TRANSACTION und COMMIT eingeklammert sind (siehe Bild 1). Die Ununterbrechbarkeit der Operationsfolge garantiert, daß aus der Sicht des Benutzers die Transaktion entweder ganz ausgeführt wird oder gar nicht. Nun kann eine Transaktion während ihres Ablaufs aus verschiedenen Gründen "steckenbleiben" oder scheitern: Verklemmung, Ausführung falscher Operationen, Erzeugung inkonsistenter Daten usw. Um sich aus solchen unerwünschten Situationen "befreien" zu können, hat die Transaktion (oder ggf. das DBS) jederzeit vor Ausführung ihrer COMMIT-Anweisung das Recht, sich zurückzusetzen, und zwar mit Hilfe von RESTORE_TRANSACTION (ABORT). Die Transaktionsdefinition impliziert weiterhin, daß alle zugehörigen Operationen unteilbar sind und logisch zusammengehören; Zwischenzustände auf der DB dürfen also nicht von anderen Transaktionen gesehen werden. Einerseits könnte nämlich ein Zwischenzustand jederzeit wieder rückgängig gemacht werden, so daß eine andere Transaktion ein "Trugbild" gesehen hätte. Andererseits kann ein solcher Zwischenzustand falsch oder unvollständig sein, da logische Konsistenz (alle spezifizierten Integritätsbedingungen sind erfüllt) nur vor Beginn und nach Ende einer Transaktion gewährleistet ist. Zusammenfassend läßt sich feststellen, daß sich beim Ablauf einer Transaktion immer nur einer von drei möglichen Ausgängen ergeben kann. Sie sind in Bild 2 veranschaulicht.

1.2 Schlüsseleigenschaften einer Transaktion

Das skizzierte Verhalten einer Transaktion läßt sich nur erzielen, wenn sie mit folgenden Eigenschaften ausgestattet wird [8]:

- **Atomicity** (Ununterbrechbarkeit): Auf der Ebene des Benutzers wird die Einhaltung der "Alles-oder-Nichts"-Eigenschaft garantiert. Das bedeutet insbesondere, daß das DBS Fehlerfälle maskieren muß; als Konsequenz daraus braucht sich der Anwendungsprogrammierer nicht mehr um das Auftreten von Fehlern zu kümmern.
- **Consistency** (Konsistenzerhaltung): Eine erfolgreiche Transaktion bewahrt die Konsistenz der Datenbank (was auch eine Vorbedingung für die Dauerhaftigkeit (Eigenschaft 4) der Ergebnisse ist). Gemäß der Definition kann sie bei COMMIT nur richtige Ergebnisse in die DB einbringen; "Richtigkeit" ergibt sich dabei aus dem Erfülltsein aller spezifizierten Integritätsbedingungen. Steht eine Aktualisierung im Widerspruch zu diesen Bedingungen, so muß die Transaktion zurückgesetzt werden.
- **Isolation** (isolierte Ausführung): Alle Aktionen innerhalb einer Transaktion müssen vor allen parallel ablaufenden Transaktionen verborgen werden, da alle Datenänderungen bis zu COMMIT nur "vorläufig" sind. Sonst würde das Rücksetzen einer Transaktion das rekursive Rücksetzen anderer nach sich ziehen. Aus diesem Grund müssen Synchronisationsmaßnahmen wie zum Beispiel Sperren [3] eingesetzt werden.

BEGIN_TA	BEGIN_TA	BEGIN_TA
READ	READ	READ
WRITE	WRITE	WRITE
•	•	•
•	•	•
•	•	•
WRITE	RESTORE_TA	
COMMIT		
normaler Ausgang	Rücksetzwunsch durch die Transaktion	vom System (Fehlerbedingung) erzwungenes Rücksetzen

Bild 2: Drei mögliche Ausgänge einer Transaktion

- **Durability** (Dauerhaftigkeit, Persistenz): Sobald die COMMIT-Operation erfolgreich abgeschlossen ist, muß das DBS die Überlebensgarantie für die in die DB eingebrachten Ergebnisse übernehmen - trotz einer Reihe von erwarteten Fehlerfällen. Das Recht der Transaktion auf Rücksetzen ist erloschen. Ebenso darf das DBS ihre Ergebnisse nicht mehr rückgängig machen, wenn beispielsweise ein Fehler in der DB entdeckt wird - in realen Systemen muß ja mit fehlerhaften Eingabedaten oder mit TAP-Fehlern gerechnet werden. In solchen Fällen gibt es nur einen Weg, da automatisches Rücksetzen das Transaktionsparadigma verletzen würde: Es muß eine sogenannte Kompensationstransaktion ausgeführt werden, um den DB-Zustand wieder mit dem der Miniwelt in Einklang zu bringen. Wohlgemerkt, damit sind solche Auswirkungen, die sich durch die Benutzung der inkorrekten Daten (in der Miniwelt und in der DB) eingestellt haben, nicht behoben. Als Konsequenz bleibt hier nur eine "manuelle" Schadensbegrenzung oder Wiederherstellung.

Diese vier Schlüsseleigenschaften - als Merkhilfe lassen sie sich als ACID-Prinzip zusammenfassen - begründen das sogenannte Transaktionsparadigma, dessen Realisierung den DBS-Entwurf stark beeinflußt hat. Es bestimmt einerseits wesentlich die internen Systemabläufe zur konsistenten Ausführung von Transaktionen und Sicherungsmaßnahmen für alle permanenten Daten; andererseits ist es Voraussetzung für die zuverlässige Programmierung von TAP. Weiterhin veranlaßt es die Maskierung aller Fehler und aller Aspekte des Mehrbenutzerbetriebes für die Anwendung, so daß einem TAP stets eine "fehlerfreie Einbenutzerumgebung vorgegaukelt" wird. Vor allem darin liegt die überragende praktische Bedeutung des Transaktionsparadigmas. Eine weiterführende Diskussion muß hier aus Platzgründen unterbleiben [4, 6, 12].

1.3 Künftige Erweiterungen des Transaktionsparadigmas

In Forschungsprojekten wird zur Zeit heftig an Erweiterungen oder Variationen des Transaktionsbegriffs gearbeitet. Wesentliche Arbeitsrichtungen sind der Entwurf von langen Transaktionen zur Kontrolle von komplexen Entwurfsaufgaben (z.B. im CAD), die Nutzung von geschachtelten Transaktionen zur sicheren und effizienten Bearbeitung von Aufgaben in verteilten Systemen sowie die Erzielung sehr hoher Transaktionsraten durch spezialisierte Transaktionssysteme.

In der wissenschaftlichen Diskussion wird heute erwartet, daß künftig der Transaktionsbegriff über sein ursprüngliches Anwendungsgebiet "DBS" hinaus eine Verallgemeinerung erfährt. Unseres Erachtens wird in nicht allzu ferner Zukunft **jede "bedeutungsvolle" Ressourcennutzung** in einem Rechensystem als Transaktion definiert. Das gilt vor allem dann, wenn eine gemeinsame Nutzung durch mehrere Anwendungen vorliegt. Dadurch werden die Vorteile der allgemeinen Transaktionseigenschaften (ACID) jeder Anwendung zuteil. Durch Erweiterung dieser Eigenschaften könnten eine Reihe von wichtigen Vorteilen für die Systemverwaltung erzielt werden. So könnte eine Transaktion beispielsweise als Einheit der Planung bei Lastkontrolle und Optimierung, zur Abrechnung von Systemleistungen oder zur Sicherstellung der Überprüfbarkeit von Vorgängen im Sinne der Revision herangezogen werden. Durch eine interne Strukturierung der Transaktion läßt sich eine Verteilbarkeit von großen Aufgaben oder eine Verknüpfung von unabhängig ausgeführten Teilaufgaben erreichen.

Aus diesen technischen Gründen ist zu erwarten, daß in künftigen Rechensystemen, die in der Regel verteilte Systeme sein werden, **nur noch Transaktionen** ausgeführt werden. Sie dienen als dynamische Kontrollstrukturen und verknüpfen zusammengehörige Berechnungen in verteilten Systemen. Dabei gewährleisten sie für den Benutzer

- eine lokale Sicht auf alle Betriebsmittel
- einen logischen Einbenutzerbetrieb, d.h. Serialisierbarkeit aller abgewickelten Transaktionen [3]
- das Verbergen von Fehlern ("Fehlerfreiheit" auf der Ebene der Programmabwicklung).

Nach unserer Einschätzung wird sich aus diesen Gründen der verallgemeinerte Transaktionsbegriff zu einem **neuen Paradigma** der Informatik entwickeln, das sowohl Programmierung als auch Entwurf großer Systeme tiefgreifend beeinflussen wird.

Diese Diskussion des Transaktionsbegriffs sollte zeigen, daß transaktionsverarbeitende Systeme künftig nicht auf den Bereich der Datenbanken beschränkt bleiben, wenngleich DB/DC-Systeme ein wichtiger oder gar der wichtigste Anwendungsbereich sind. Sie spielen wohl auch die Vorreiterrolle bei der Entwicklung großer Systeme mit den Schlüsseleigenschaften "hohe Leistungsfähigkeit, hohe Verfügbarkeit und modulare Erweiterbarkeit". Da die Konzepte und Verfahren wegen der Notwendigkeit, alle Operationen auf großen Datenvolumina zu synchronisieren, einen hohen Grad an Allgemeingültigkeit aufweisen müssen, ist zu erwarten, daß sie auch in anderen Bereichen eingesetzt werden können.

2. Ein DB/DC-Systemmodell

Welche Ansprüche stellt nun das Transaktionskonzept an die Fehlertoleranz eines Transaktionssystems? Um sich die daraus ergebenden Anforderungen und Ziele genauer erörtern zu können, führen wir ein vereinfachtes Systemmodell ein. Dabei genügt es, daß wir uns auf den Systemkern konzentrieren, weil er allein die geforderte Fehlertoleranz zu verkörpern hat: er besteht aus dem DBS sowie dem Transaktionsmonitor (TP-Monitor, DC-System) und wird als DB/DC-System bezeichnet.

In Bild 3 ist der Aufbau eines Transaktionssystems skizziert. Die oberste Schicht ist für alle Aufgaben der Ein-/Ausgabe zu den Terminals verantwortlich. Zur Sicherstellung der Wiederholbarkeit der Transaktionsverarbeitung und der Ergebnisausgabe führt sie eine Protokollierung aller Ein- und Ausgabenachrichten durch. Im Grunde genommen umfaßt diese Schicht Kommunikations-Software, die gewöhnlich dem Betriebssystem (BS) zugerechnet wird. Schicht 2 wird gebildet von der Menge der Anwendungsprogramme (TAPs), die quasi sandwich-artig vom TP-Monitor umschlossen werden. Schicht 3 ist dabei vor allem verantwortlich für die Koordination des Mehrbenutzer-Betriebs (Multi-Tasking [7]) und für die Abwicklung der DB- und DBS-Aufrufe.

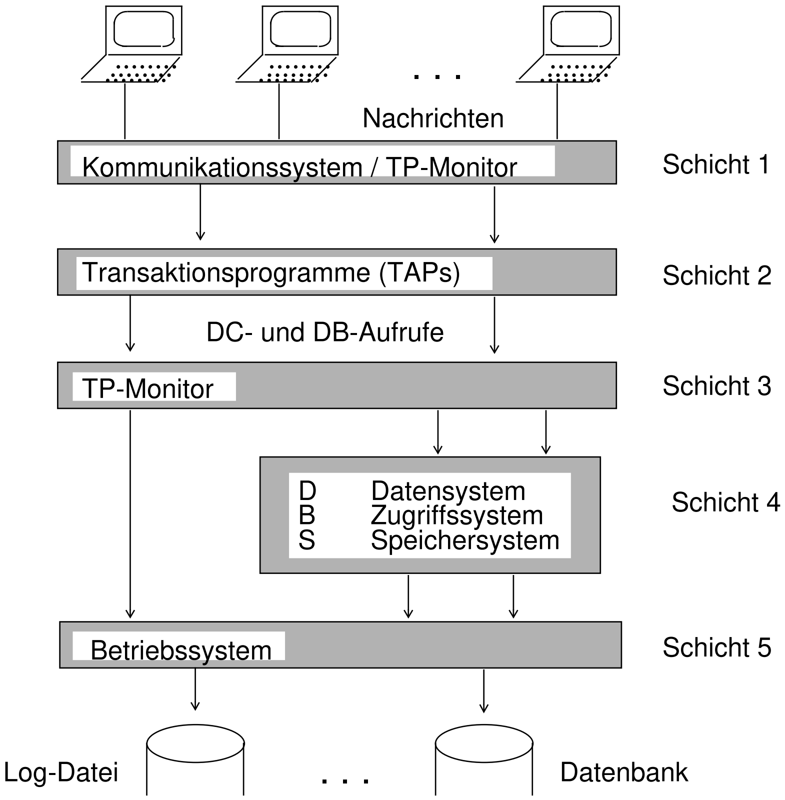

Bild 3: Schichtenmodell eines Transaktionssystems

Das DBS ist selbst ein komplexes System mit mehreren Abbildungsschichten [6], wie in Bild 3 angedeutet. Diese detaillierte Betrachtung ist in unserem Zusammenhang unwesentlich. Für die Fehlerbehandlung ist jedoch folgende Tatsache sehr bedeutsam: Aus Performance-Gründen verwaltet das DBS einen DB-Puffer beträchtlicher Größe im (flüchtigen) Hauptspeicher. Mit der Verfügbarkeit billiger Speicher wächst heute die typischen Pufferkapazität schnell (1-50 MBytes). Im Grenzfall wird sogar die gesamte DB im Systempuffer gehalten. Die Ersetzung der Seiten (auch der geänderten) im DB-Puffer erfolgt nach Bedarf und wird durch eine Ersetzungsverfahren (z.B. LRU) gesteuert; sie werden also nicht unmittelbar nach Änderung oder bei Transaktionsende ausgeschrieben. Für das Ausschreiben von Seiten auf Externspeicher benutzt das DBS (wie der TP-Monitor) die Dienste des BS, das ein Datensystem zur Verfügung stellt.

Bild 3 läßt auch den prinzipiellen Weg der Transaktionsverarbeitung (Aufrufhierarchie) erkennen. Eine Terminalnachricht (Bildschirmmaske) wird vom DC-System (über das BS) empfangen; das darin angesprochene TAP wird initialisiert, erhält seine aktuellen Parameter (EXTRACT) und führt seine Berechnungen aus. Alle Aufrufe an das DBS (die DB-Transaktion, siehe Bild 1) werden von der DC-Komponente registriert, um im Fehlerfall gewisse Recovery-Maßnahmen treffen zu können. DBS-seitig werden die Transaktionseigenschaften garantiert - aber natürlich nur für die DB-Transaktion. Es ist offensichtlich, daß aus Benutzersicht der gesamte TAP-Ablauf (Bild 1) als ununterbrechbare Einheit zusammengefaßt werden sollte. Später zeigen wir, wie dies zu bewerkstelligen ist (eine gemeinsame DC- und DB-Transaktion).

3. Aufgaben der DB/DC-Fehlerbehandlung

Die Transaktionseigenschaften definieren nur eine fehlerfreie Ablaufumgebung für TAP (aus der Sicht des Benutzers) - sie garantieren jedoch nicht automatisch einen zeitlich ununterbrochenen Systembetrieb. Hohe Verfügbarkeit oder Ausfallsicherheit setzen deshalb noch wesentlich mehr voraus als die Einhaltung des Transaktionskonzeptes.

Bei der Diskussion von Fehlertoleranzmaßnahmen im Rahmen der Transaktionsverarbeitung müssen wir uns lediglich mit zwei der vier ACID-Eigenschaften auseinandersetzen. Atomarität fordert nur die Maskierung von Fehlern (z.B. Systemausfall), nicht jedoch ihre Verhinderung, und Dauerhaftigkeit impliziert nur das Überleben aller bei COMMIT freigegebenen Daten. Ein Gerätefehler kann durchaus zu einer längeren Nichtverfügbarkeit der Daten führen, ohne das diese Eigenschaft verletzt wird.

Um die Aufgaben der DB/DC-Recovery festlegen zu können, müssen drei Fragen genau beantwortet werden:

- Welche Fehlertypen werden erwartet und sind automatisch durch das System zu behandeln?
- Unter welchen Voraussetzungen hat die Fehlerbehandlung Erfolg?
- Welcher Zielzustand muß nach erfolgreicher Recovery erreicht sein?

3.1 Welche Fehler werden erwartet?

Eine ausführliche Diskussion der Fehlerproblematik findet sich in [5]. Hier charakterisieren wir nur die Fehler, für die typischerweise Recovery-Maßnahmen vorgesehen sind:

- Transaktionsfehler: Eine Transaktion gerät in Konflikt mit anderen Transaktionen (Verklemmung) oder gefährdet die Systemkonsistenz (Verletzung von Integritätsbedingungen, zu großer Ressourcenverbrauch, Programmfehler usw.). Zur Bereinigung der Situation wird sie vom DBS zurückgesetzt. Der Rücksetzwunsch der Transaktion (ABORT) wird technisch in gleicher Weise behandelt.
- Systemfehler: Sie werden ausgelöst durch Fehler in der Hardware oder in wichtigen Systemkomponenten und führen zum Systemausfall, d.h. zu einer unkontrollierten Beendigung der gesamten Verarbeitung. Der aktuelle Systemzustand - insbesondere auch der DB-Pufferinhalt - ist verlorengegangen.
- Gerätefehler: Im laufenden Betrieb wird entdeckt, daß ein Block, eine Spur oder ein Zylinder nicht mehr gelesen werden kann oder daß gar die ganze Magnetplatte zerstört ist.
- Kommunikationsfehler: Nachrichtenverbindungen können gestört sein, was zum Verlust, zur Verfälschung, zur Verzögerung oder zu einer falschen Zustellungsreihenfolge der Nachrichten führen kann.

Sind DBS oder DB/DC-System verteilt, dann wird auch noch der Ausfall einer Knotens als erwarteter Fehler angenommen, wobei das Restsystem "ungestört" weiterlaufen kann. Andere Arten von Fehlern, die ggf. auftreten, sind "nicht-erwartete Fehler". Solche Fehler entziehen sich der automatisierten Recovery, da keine Vorsorgemaßnahmen getroffen wurden. Je nach Schwere des Fehlers kann man hierbei von "Katastrophen" sprechen. Manuelle Maßnahmen erreichen oft nicht die Konsistenzanforderungen, wie sie für die automatisierten Verfahren eingeplant sind [12].

3.2 Voraussetzungen für die Fehlerbehandlung

Algorithmen zur Fehlerbehandlung "reparieren" einen als fehlerhaft erkannten Systemzustand. Die allgemeine Problematik der Fehlererkennung wollen wir hier nicht diskutieren [10] - bei den oben eingeführten Fehlerklassen ist sie in der Regel offensichtlich. Die grundlegenden Annahmen für eine erfolgreiche Fehlerbehandlung tragen dazu bei, die Fehlererkennung einfach und sicher zu gestalten. In heutigen DB/DC-Systemen setzen alle Recovery-Maßnahmen typischerweise folgende Systemeigenschaften voraus:

- Externspeichermedien, welche die DB und Datensicherungsinformationen aufnehmen, verhalten sich "quasi-stabil" [11]. Damit wird gefordert, daß korrekte Schreib- und Leseoperationen verfügbar sind und daß "umgekippte Bits" durch Paritätsprüfung erkannt werden.
- Der gesamte DBS-Code wird als fehlerfrei angenommen (was bei der Komplexität dieser Systeme natürlich fragwürdig ist und durch die Praxis ständig widerlegt wird).
- Die zu Sicherungszwecken gesammelten Daten (Log) sind korrekt und im Fehlerfall einsetzbar.
- Fehlersituationen sind voneinander unabhängig; insbesondere gehen alle Recovery-Verfahren davon aus, daß nur ein Fehler auf einmal auftritt.
- Der Einfachheit halber nehmen wir an, daß das DB/DC-System sich auf die Korrektheit der Nachrichten und der Zuverlässigkeit der Übertragungswege verlassen kann.

3.3 Transaktionsorientierte Recovery-Maßnahmen

Für alle erwarteten Fehler müssen systemseitig Recovery-Maßnahmen zur Verfügung gestellt werden, die unter den skizzierten Voraussetzungen eine erfolgreiche Fehlerbehandlung erlauben. Dabei hat das DBS den Löwenanteil zu erbringen, da die meisten und schwierigsten Fehler die DB betreffen. Der Systemzustand, in dem ein Recovery-Algorithmus gestartet wird, hängt stark vom Fehlertyp ab und muß beim Entwurf der Reparaturmaßnahme berücksichtigt werden. Das Ziel einer erfolgreichen Recovery ist jedoch notwendigerweise stets dasselbe. Die "Alles-oder-Nichts"-Eigenschaft (zusammen mit der Dauerhaftigkeit) der Transaktion verlangt, daß die Ergebnisse aller beendeten Transaktionen vollständig in der DB erhalten bleiben, während die aller gescheiterten Transaktionen spurlos zu entfernen sind. Diesen Zielzustand bezeichnen wir als "jüngsten transaktionskonsistenten DB-Zustand". Dazu stellt ein DBS folgende Maßnahmen bereit:

- Transaktionsfehler-Recovery: Im laufenden DB-Betrieb wird eine Transaktion ohne Rückwirkung auf andere zurückgesetzt (transaction UNDO). Sie muß sehr effizient implementiert sein (Rücksetzdauer von msec oder wenigen sec).
- Systemfehler-Recovery: Nach einem Systemausfall sind i.a. große Mengen geänderter Daten im Systempuffer verlorengegangen. Nur die auf nicht-flüchtigen Externspeichern befindlichen Daten sind verfügbar (in einem Zustand, der von der Einbringstrategie abhängt). Es müssen trotzdem die Ergebnisse aller bis dahin beendeten Transaktionen - soweit noch nicht vorher geschehen - in die DB eingebracht werden (partial REDO). Wenn Daten unbeendeter Transaktionen (dirty data) in die DB gelangt sind (beispielsweise durch die Seitenersetzung im DB-Puffer), so müssen diese aus der DB entfernt werden (global UNDO). Je nach Systemstabilität ist einmal pro Woche/Monat mit dem Auftreten eines solchen Fehlers zu rechnen. Die DB-seitige Recovery-Zeit sollte im Sekundenbereich oder bei wenigen Minuten liegen.
- Gerätefehler-Recovery: Hierbei ist zu beachten, daß der Gerätefehler (z.B. zerstörte Plattenspur) oft schon längere Zeit existiert, nur noch nicht bemerkt wurde. In jedem Fall ist der Inhalt des zerstörten Bereiches zu rekonstruieren (Media-Recovery). Glücklicherweise sind solche Fehler relativ selten - pro Platte kann man heute eine Zeit von bis zu 10 Jahren zwischen dem Auftreten eines solchen Fehlers erwarten. Wenn die DB jedoch 100 Platten oder mehr umfaßt, gibt es durchaus eine "tägliche Bedrohung". Als Recovery-Zeit müssen oft Stunden angesetzt werden.

In einem DB/DC-System muß zusätzlich noch etwas gegen Kommunikationsfehler getan werden. Wir unterstellen, daß Übertragungswege und -protokolle im Verantwortungsbereich des BS liegen. Der TP-Monitor sorgt dann nur für eine

- Nachrichten-Recovery: Eine ankommende Nachricht wird gesichert, damit das entsprechende TAP ggf. nach einem Systemfehler ohne Mitwirkung des Benutzers erneut gestartet werden kann. Geht eine Ausgabe-Nachricht verloren, so muß sie wiederholt werden können.

4. Sicherungsmaßnahmen im Normalbetrieb

Jede Art der Fehlerbehandlung setzt geeignete Redundanz im Systemablauf oder als Sicherungsinformation voraus. Die Sicherungsvorkehrungen in einem DB/DC-System haben also die Aufgabe, während des normalen Betriebs redundante Mechanismen bereitzustellen oder hinreichend viele Informationen zu sammeln, daß nach einem erwarteten Fehler eine automatische Recovery erfolgen kann. Dafür haben sich zwei Verfahrensklassen herausgebildet: "Forward Recovery" und "Backward Recovery" [9]. Aus der ersten Klasse gibt es im DB-Kontext bestenfalls teure Speziallösungen für einzelne Fehlertypen. Gerätefehler lassen sich beispielsweise mit Hilfe des Prinzips der fehlertoleranten Speicherung (z.B. Spiegelplatten) unterbrechungsfrei behandeln, wobei der DB-Betrieb "nach vorne" weitergeführt werden kann. Eine fehlertolerante Ausführung läßt sich mit Hilfe des Konzepts der Prozeß-Paare [2] in Mehrprozessor-Systemen erzielen: ein aktiver PRIMARY-Prozeß informiert lückenlos (und mit hoher Frequenz) einen passiven BACKUP-Prozeß, so daß im Falle des PRIMARY-Ausfalls die unterbrechungsfreie Fortführung der anwendungsbezogenen Aktivitäten bewerkstelligt werden kann. Prinzipiell könnte man natürlich fragen, ob Forward Recovery als Konzept nicht ausreicht, da sogar ein unterbrechungsfreier Betrieb gewährleistet wird. Generell muß jedoch gesagt werden, daß dieses Konzept fast prohibitive Kosten verursacht - also bestenfalls selektiv einzusetzen ist. Es nutzt auch überhaupt nicht die "Freiheiten des Transaktionskonzeptes" aus, wodurch Rücksetzung und Wiederholung von Aktionsfolgen eingeräumt werden. In jedem Fall würden zusätzliche Maßnahmen erforderlich, da das Recht auf ABORT die Notwendigkeit des anwendungsseitigen Zurücksetzens impliziert. Aus diesen Gründen dient die Backward Recovery als allgemeines Konzept, wobei sie sich bei Rücksetzungen und Wiederholungen ausschließlich an Transaktionsgrenzen orientiert.

4.1 Wie wird Sicherungsinformation gesammelt?

Da die Transaktion die Einheit der Recovery ist, muß die Sicherungsinformation (UNDO/REDO-Information) auch transaktionsbezogen gesammelt werden. Sobald eine Transaktion erfolgreich beendet ist und ihre Änderungen in die DB eingebracht sind oder eine gescheiterte Transaktion auch in der DB zurückgesetzt ist, existiert sie im laufenden Betrieb nicht mehr - also kann sie nicht mehr "Opfer" von Transaktions-, System- und Kommunikationsfehlern werden. Jedoch können ihre Ergebnisse noch sehr viel später durch einen Gerätefehler zerstört werden. Aus diesen Gründen hat sich in der Praxis eine Zweiteilung durchgesetzt. Für die kurzfristige Recovery werden UNDO- und REDO-Informationen in einer temporären Protokolldatei (Log-Datei) gesammelt. Nachrichtenprotokolle können in derselben oder in einer speziellen Datei des TP-Monitors gespeichert werden. Nach Abschluß einer Transaktion (im oben skizzierten Sinne) wird diese Information überflüssig. Zur Gerätefehlerbehandlung ist eine langfristige Recovery einzuplanen. Dazu müssen eine oder mehrere (transaktionskonsistente) Archivkopien der DB (nach dem Generationsprinzip) gehalten werden. Zusätzlich wird REDO-Information für alle erfolgreichen Transaktionen in einer Archiv-Protokolldatei (Archiv-Log) gesammelt, um die Auswirkungen von weit zurückreichenden Gerätefehlern durch Wiederholung der Transaktionsergebnisse beheben zu können. In der Praxis schützt man oft die Sicherungsdaten selbst durch Redundanz, in dem man eine doppelte Aufzeichnung der Log-Daten (Duplex-Log) vornimmt.

4.2 Protokollierung in Datenbanksystemen

Prinzipiell lassen sich zwei Arten von Protokollierungstechniken (Logging) unterscheiden:

- Physische Protokollierung: Die alten/neuen Zustände der Objekte werden auf die Log-Datei geschrieben. Objekte sind dabei in der Regel Seiten oder Einträge.
- Logische Protokollierung: Die Änderungs-DML-Befehle zusammen mit ihren Parametern werden auf die Log-Datei geschrieben.

Es ist klar, daß nicht jeder Log-Eintrag separat auf die Log-Datei ausgegeben werden muß. Vielmehr werden aus Leistungsgründen Log-Einträge in einem entsprechenden Puffer gesammelt und dann seitenweise oder gar in größeren Einheiten (bei chained I/O) ausgeschrieben. Dabei sind jedoch drei für die DB/DC-Recovery fundamentale Prinzipien zum beachten:

1. WAL-Prinzip: Wird eine Änderung einer unvollständigen Transaktion (dirty data) in die DB eingebracht, so ist vorher die zugehörige UNDO-Information auf die Log-Datei zu schreiben (Write-Ahead-Log).
2. COMMIT-Regel: Spätestens mit der COMMIT-Verarbeitung einer Transaktion müssen ihre REDO-Informationen auf der Log-Datei (nicht jedoch ihre geänderten Daten) gesichert sein.

Diese Regeln besagen, daß ein Systemausfall zu jeder Zeit erwartet werden muß. Deshalb sind irreversible Operationen abzusichern, d.h., Rücksetzbarkeit bzw. Wiederholbarkeit sind zu gewährleisten. Analog gilt für das DC-System die Regel:

3. Nachrichten-Logging: Nach Eingabe und vor Ausgabe einer Terminal-Nachricht ist ein Log-Eintrag mit der entsprechenden Nachricht zu sichern.

Die Art der Protokollierung beeinflußt das Leistungsverhalten eines DB/DC-Systems erheblich. Für unsere Beispieltransaktion in Bild 1 ist für das Sammeln von UNDO- und REDO-Information DB-seitig mit folgendem Aufwand zu rechnen:

- bei physischer Seitenprotokollierung (inkl. Systemdaten) mit 12 und mehr Schreibvorgängen
- bei physischer Eintragsprotokollierung (ca. 500 Bytes) mit einer Log-Ausgabe
- bei logischer Protokollierung ebenfalls nur mit einem Schreibvorgang.

Zusätzlich würden noch zwei Ausgaben durch das Nachrichten-Logging anfallen. Als Optimierungsgrundsatz sollte in jedem Fall die Minimierung der Log-E/A gelten. Benutzen TP-Monitor und DBS eine gemeinsame Log-Komponente, so können die DB- und DC-seitigen Log-Informationen zusammen gepuffert und ausgeschrieben werden. Eine weitere Möglichkeit, Schreibvorgänge einzusparen, ergibt sich durch das Konzept des "Group Commit": Dabei werden Transaktionen bei COMMIT etwas verzögert, damit ihre REDO-Informationen nach Regel 2 gemeinsam ausgeschrieben werden können. Auf diese Weise läßt sich ein mittlerer Bedarf von weniger als einer E/A für alle Log-Daten einer Transaktion erreichen.

4.3 Nutzung nicht-unterbrechbarer Einbringstrategien

Bisher wurde unterstellt, daß eine Seite immer wieder auf den gleichen Block der Platte zurückgeschrieben wird (update-in-place). Eine solche Strategie, die man typischerweise in heutigen DBS vorfindet, wird auch als NOTATOMIC bezeichnet, weil das Rückschreiben von n Seiten zu beliebigen Zeitpunkten durch einen Systemausfall unterbrochen werden kann. In Forschungsprojekten wurden in jüngerer Zeit Verfahren vorgeschlagen, die ein Einbringen von n Seiten in ununterbrechbarer Weise gestatten (ATOMIC-Strategien). Das Schattenspeicher- und das zusätzliche Zusatzdatei-Konzept besitzen beispielsweise diese Eigenschaft [6]. ATOMIC-Strategien frieren im Prinzip den DB-Zustand auf der Platte ein und sammeln DB-Änderungen in einem separaten Bereich. Durch "Umschalten" können diese Änderungen atomar in die DB eingebracht werden - es wird ein neuer Sicherungspunkt erzeugt. Nach einem Systemfehler wird also stets eine definierter Zustand vorgefunden - der alte oder der neue DB-Zustand.

Solche Verfahren bauen im Prinzip Redundanz in die Abbildungsmechanismen der Speicherungsstrukturen ein, die für die Recovery vorteilhaft ausgenutzt werden kann. Sie belasten jedoch den Normalbetrieb sehr, da die Wartung der Redundanz viele zusätzliche E/A-Operationen kostet. Diese Verfahren werden in der Regel kombiniert mit Log-Verfahren eingesetzt, wobei die Optimierung einiger nur auf einen wesentlichen Zusammenhang hingewiesen werden:

- Einschränkung der logischen Protokollierung: Logische Protokolle können nur mit ATOMIC-Strategien kombiniert werden.

Sie setzen voraus, daß die DB nach einem Systemausfall in einem konsistenten Zustand ist, auf dem die protokollierten DML-Befehle (und ihre Umkehroperationen) abgewickelt werden können. Physische Protokollierung kann mit allen Einbringstrategien kombiniert werden; sie wird in den meisten DBS eingesetzt.

4.4 Welche Maßnahmen erfordert COMMIT?

Die COMMIT-Verarbeitung ist sehr komplex, da trotz der Möglichkeit eines Systemausfalls in eindeutiger Weise über das "Schicksal" einer Transaktion entschieden werden muß. Sie läuft prinzipiell in zwei Phasen ab [Gr78]:

- Zwei-Phasen-COMMIT bei zentralisierten DBS: In einer ersten Phase wird die Wiederholbarkeit der Transaktion sichergestellt, indem die erforderlichen REDO-Informationen zusammen mit einem COMMIT-Eintrag auf die Log-Datei geschrieben werden. In Phase 2 werden die Betriebsmittel (Sperren und Daten) freigegeben sowie die Meldung des erfolgreichen Abschlusses an den Benutzer geschickt.

Mit dem Absenden der COMMIT-Anweisung an das DBS hat das TAP sein Recht auf einseitiges Zurücksetzen aufgegeben. Das DBS kann jetzt nur noch durch einen Systemausfall unterbrochen werden. Wird nach einem solchen der COMMIT-Eintrag in der Log-Datei gefunden, so überlebt die Transaktion. Im anderen Fall wird sie zurückgesetzt.

Durch das beschriebene Verfahren erreichen wir jedoch nur die Atomarität der DB-Transaktion. Aus der Sicht des Terminalbenutzers sollten jedoch DC- und DB-Transaktion zu einer ununterbrechbaren Einheit zusammengeschweißt werden [9]. Das folgende Protokoll löst dieses Problem:

- Zwei-Phasen-COMMIT in einem DB/DC-System: Der TP-Monitor sendet anstelle des COMMIT ein PREPARE-TO-COMMIT an das DBS. Es führt nur Phase 1 aus und stellt damit die Wiederholbarkeit der Transaktion her. Der TP-Monitor bekommt mit einem O.K. die Kontrolle zurück und kann sein DC-COMMIT abwickeln. Ist dieses erfolgreich, wird das DBS mit COMMIT aufgefordert, die Phase 2 durchzuführen.

Durch dieses Protokoll entscheidet letztlich der TP-Monitor als Master über den Ausgang einer Transaktion, das DBS befolgt nur die Anweisungen. Nach erfolgreichem Sichern der Ausgabe-Nachricht (END-DC in Bild 1) ist der "point of no return" erreicht, so daß dann dem DBS der erfolgreiche Abschluß signalisiert werden kann (Phase 2). Es sei nur noch angemerkt, daß dieses Protokoll in einfacher Weise für den Einsatz in verteilten DB/DC-Systemen verallgemeinert werden kann.

5. Techniken der transaktionsorientierten Recovery

5.1 Transaktions- und Systemfehler-Recovery

Wegen ihrer Effizienzanforderungen und der Häufigkeit ihres Einsatzes sind die Recovery-Verfahren (und die zugehörigen Vorsorgemaßnahmen) für Transaktions- und Systemfehler besonders wichtig für das Systemverhalten. Neben der gewählten Einbringstrategie (ATOMIC/NOTATOMIC) üben bestimmte Eigenschaften der Seitenersetzungsstrategie und der COMMIT-Behandlung der Seiten im DB-Puffer einen entscheidenden Einfluß auf die Art der Recovery-Maßnahmen aus. Bei der Seitenersetzung wird unterschieden zwischen

- STEAL: Geänderte Seiten können jederzeit - insbesonders vor COMMIT der ändernden Transaktion - in die DB eingebracht werden.
- NOSTEAL: Vor COMMIT werden keine geänderten Seiten eingebracht.

Die COMMIT-Behandlung läßt folgende Alternative zu:

- FORCE: Geänderte Seiten sind spätesten in Phase 1 von COMMIT in die DB einzubringen.
- NOFORCE: Einbringen von Änderungen wird bei COMMIT nicht erzwungen.

Die Eigenschaften der Einbringstrategie, der Seitenersetzung und der COMMIT-Behandlung lassen sich nun so kombinieren, daß das in Bild 4 gezeigte Klassifikationsschema [8] herauskommt, das im Prinzip alle Realisierungskonzepte für die diskutierten Recovery-Verfahren angibt. Um den Bezug zur Praxis herzustellen, haben wir einige existierende DBS an die entsprechenden Klassifikationspfade geschrieben.

In Kurzform seinen noch einige wesentliche Eigenschaften der Recovery für die gewählten Klassifikationskriterien aufgezeigt. NOSTEAL impliziert, daß die Transaktionsfehler-Recovery ausschließlich im Hauptspeicher (DB-Puffer) ablaufen kann - also sehr schnell ist. Auch bei einem Systemausfall sind keine UNDO-Operationen auf der DB erforderlich. Bei STEAL dagegen können für beide Fehlertypen UNDO-Operationen auf der DB (transaction UNDO und global UNDO) erforderlich sein, wenn Änderungen zurückzusetzender Transaktionen in die DB gelangt sind. FORCE impliziert, daß keine REDO-Operationen nach einem Systemausfall notwendig sind, da alle Änderungen einer Transaktion die DB spätestens in Phase 1 von COMMIT erreichen (kein partial REDO). Erkauft wird diese wünschenswerte Eigenschaft durch hohen Schreibaufwand bei COMMIT, verlängerte Antwortzeiten und schlechte Nutzung großer DB-Puffer. NOFORCE übt keinen Schreibzwang aus, was bedeutet, daß sich bestimmte geänderte Seiten (sogenannte Hot Spots u.a.) noch sehr lange nach dem ersten (und weiteren) COMMIT im DB-Puffer aufhalten können. Offensichtlich erzwingt ein Systemausfall die Wiederholung aller Änderungen für solche Seiten (partial REDO). Um den Recovery-Aufwand für NOFORCE-Strategien zu begrenzen, müssen sie im praktischen Einsatz mit sogenannten Sicherungspunkt-Verfahren (checkpoints [8]) kombiniert werden, die bei einem Systemausfall einen definierten Aufsetzpunkt für das "partial REDO" liefern.

Abschließend soll eine Empfehlung für eine Verfahrensklasse gegeben werden. Dabei fließen folgende Annahmen ein: Systemausfälle sind nicht allzu häufig, und der Normalbetrieb sollte nicht zugunsten der Systemfehler-Recovery belastet werden. Das ist eine optimistische Sichtweise, bei der im Fehlerfall ggf. etwas höhere Kosten anfallen. Dann sollte als Einbringstrategie NOTATOMIC (wie in fast allen existier-

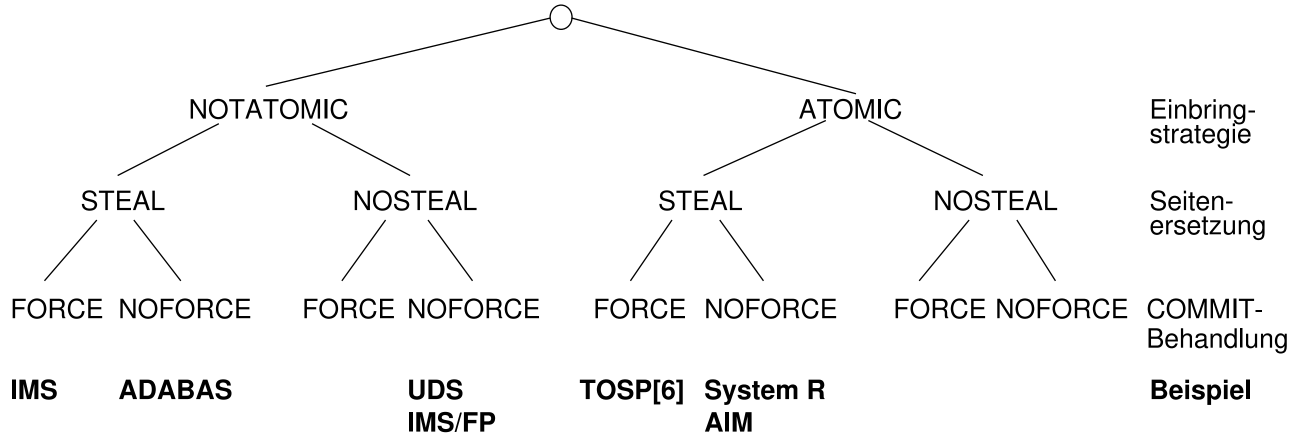

Bild 4: Klassifizierungsschema für Recovery-Konzepte

enden DBS) gewählt werden. Immer größer werdende DB-Puffer sollten eine Realisierung von <u>NOSTEAL</u> ohne Probleme erlauben (keine UNDO-Operationen auf der DB). Schließlich belastet FORCE die Antwortzeit der Transaktion sehr stark: im Beispiel aus Bild 1 müßten inkl. von Änderungen in den Systemdaten 6 und mehr Seiten bei COMMIT in die DB geschrieben werden! Deshalb ist einer <u>NO-FORCE</u>-Strategie in jedem Fall der Vorzug zu geben, wobei allerdings ein geeignetes Sicherungspunkt-Verfahren den REDO-Aufwand zu begrenzen hat.

5.2 Gerätefehler- und Nachrichten-Recovery

Wie schon erwähnt, spielt die Media-Recovery eine Sonderrolle. Das Spektrum der Techniken soll hier nicht vertieft werden [12] - sie beruht ausschließlich auf Archivkopien und den REDO-Informationen aller erfolgreichen Transaktionen seit dem Erstellzeitpunkt der für die Recovery benutzten Archivkopie. In Systemen hoher Verfügbarkeit sind solche Maßnahmen untauglich, weil ggf. bestimmte Daten mehrere Stunden lang unzugänglich sind. Deshalb muß dieser Fehlertyp vermieden werden durch fehlertolerante Speicher mit Replikation der Daten auf Geräten mit unabhängigen Fehlermodi. Durch Einsatz von Spiegelplatten läßt sich ein sehr zuverlässiger fehlertoleranter Speicher nachbilden (> 1000 Jahre MTBF für ein Plattenpaar). Trotzdem werden in der Praxis zur Recovery-Vorsorge zusätzlich noch Archiv-Kopien und -Logdateien gehalten.

Noch eine letzte Bemerkung zur Nachrichten-Recovery, die unter Verantwortung des TP-Monitors die DB-seitigen Maßnahmen ergänzt. Da die Eingabe-Nachrichten gesichert sind, ist es möglich, bei Transaktions- und Systemfehler zurückgesetzte Transaktionen erneut zu starten. Bei Einschritt-Transaktionen können solche Fehler prinzipiell maskiert werden, bei Mehrschritt-Transaktionen wird der Benutzer nur involviert, wenn bei der Wiederholung andere Ergebnisse als beim ersten Durchlauf erzielt werden [9]. Erreicht eine Ausgabe-Nachricht das Terminal nicht, so kann sie nach Behebung der Kommunikationsprobleme erneut gesendet werden. Damit ist es möglich, ihre Zustellung zu garantieren - nicht jedoch ihre einmalige Zustellung. Man denke hier an einen Bankautomaten als Transaktionssystem.

Literaturverzeichnis

[1] Anon et. al.: A prototype for a highly available database system, IBM Research Report RJ 3755, San Jose, Calif., Jan. 1983.

[2] Bartlett, F.J.: A "non stop" operating system, Tandem Computers Inc, Cupertino, 1977.

[3] Eswaran, K.P., Chamberlin, D.D.: The notions of consistency and predicate locks in a data base system, Comm. ACM, 19:11, 1976, pp. 624-633.

[4] Gray, J.: Notes on database operation systems, in: Operation Systems: an Advanced Course, Springer-Verlag, LNCS 60, 1978, S. 393-481.

[5] Gray, J.N., et. al.: The recovery manager of the System R database manager, ACM Computing Surv. 13:2, 1981, pp. 223-242.

[6] Härder, T.: Realisierung von operationalen Schnittstellen, in: Datenbank-Handbuch, Springer-Verlag, 1987, S. 163-335.

[7] Härder, T., Meyer-Wegener, K.: Transaktionsverarbeitung und TP-Monitore - Eine Systematik ihrer Aufgabenstellung und Implementierung, in: Informatik Forschung und Entwicklung, Bd. 1 (1986), Heft 1, S. 3-25.

[8] Härder, T., Reuter, A.: Principles of transaction-oriented database recovery, in: ACM Computing Surv. 15:4, 1983, S. 287-317.

[9] Meyer-Wegener, K.: Transaktionssysteme - Funktionsumfang, Realisierungsmöglichkeiten, Leistungsverhalten, B.G. Teubner, 1987.

[10] Randell, B., Lee, P.A., Treleaven, P.C.: Reliability issues in computing system design, ACM Computing Surv. 10:2, 1978, pp. 123-165.

[11] Reuter, A.: Fehlerbehandlung in Datenbanksystemen, Carl Hanser, München, 1981.

[12] Reuter, A.: Maßnahmen zur Wahrung von Sicherheits- und Integritätsbedingugen, in: Datenbank-Handbuch, Springer-Verlag, 1987, S. 337-479.