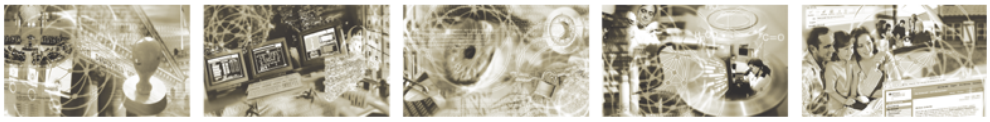
Bundesamt
für Sicherheit in der
Informationstechnik



Band B, Kapitel 8: Datenbanken

Im Umfeld der Hochverfügbarkeit

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 22899 9582-0
E-Mail: hochverfügbarkeit@bsi.bund.de
Internet: https://www.bsi.bund.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Datenbanken im HV-Umfeld

Die Anforderungen, die Geschäftsprozesse an die Verfügbarkeit von IT-Systemen stellen, sind aufgrund der engen Verzahnung betrieblicher und technischer Prozesse in den letzten Jahren immer weiter gestiegen. Nahezu alle heutigen Geschäfts- oder IT-Prozesse, z. B. im Rahmen der Nutzung von JiT, B2B, B2C, B2E oder Web2.0, basieren auf hochverfügbaren IT-Systemen. Eine Kernkomponente bei allen zugrunde liegenden IT-Prozessen bildet das Datenbanksystem (DBS). In der Historie haben sich verschiedene DB-Modelle entwickelt, u. a. relationale, hierarchische und objektorientierte Datenbanken. Das Modell mit der größten Verbreitung ist das relationale Datenbankmodell. Aus diesem Grund beschränken sich die Ausführungen in diesem Beitrag auf relationale Datenbanken.

Aktuell erhältliche DBS müssen in der Lage sein, alle für die Geschäftsprozesse notwendigen Informationen bedarfsgerecht zu liefern. Ist dies nicht der Fall, so sind unter Umständen Geschäfts- oder IT-Prozesse gefährdet. Damit die Verfügbarkeit der Prozesse sichergestellt wird, ist es erforderlich, über die komplette Prozesskette hinweg eine adäquate Verfügbarkeit der Informationen zu gewährleisten. Hierzu werden in dem DBS alle, für den Gesamtprozess erforderlichen Daten vorgehalten und der Zugriff auf sie ermöglicht. Stehen einem Nachfrager einer Information (Anwender oder Anwendung) die Informationen nicht bedarfsgerecht zur Verfügung, so ist keine Hochverfügbarkeit (HV) gegeben. Die Gesamtverfügbarkeit der DBS ergibt sich dabei aus den Verfügbarkeiten aller beteiligter Komponenten und Dienste. So stellt neben der Verfügbarkeit der Datenbank und des DBMS auch die Verfügbarkeit der DB-Dienste eine wichtige Voraussetzung zur Erreichung einer hohen Verfügbarkeit dar. Die Betrachtung erfolgt dabei differenziert auf unterschiedlichen Ebenen. Die veranschaulicht dies und stellt die in diesem Beitrag verwendeten Begrifflichkeiten im Zusammenhang dar.

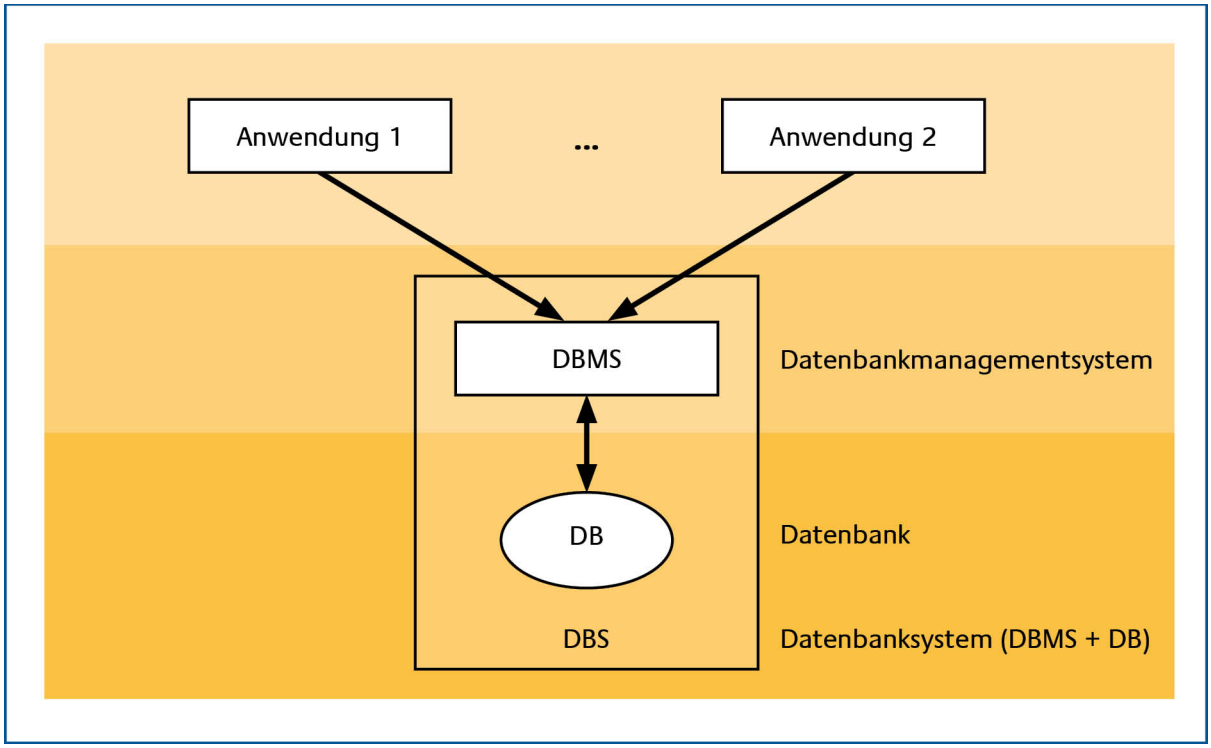


Abbildung 1: Definition der Begrifflichkeiten

So können - beginnend bei allgemeinen Konzepten zur redundanten Datenverteilung, über Maßnahmen auf der Transaktionsebene bis hin zu unterschiedlichen Verfahren beim DB-Backup und -Recovery - viele Mittel eingesetzt werden, die die Verfügbarkeit der Datenbank und damit des Gesamtsystems entscheidend verbessern können. Bei allen Maßnahmen zur Realisierung hoher Verfügbarkeit des DBS ist aber auch zu bedenken, dass diese immer im Kontext der Gesamtinfrastruktur zu betrachten sind. Maßnahmen zur Verbesserung der Verfügbarkeit können, wenn sie falsch implementiert werden, auch das Gegenteil bewirken und vorher nicht bedachte negative Folgeeffekte verursachen.

Bei der Planung von Maßnahmen zur Verbesserung der Verfügbarkeit sollte auch beachtet werden, dass Datenbanken häufig als Teilbausteine unterschiedlicher Anwendungen fungieren. Deshalb gibt es hinsichtlich der Anforderungen an die Verfügbarkeit der Daten und Datenbankdienste - in Abhängigkeit der jeweiligen Anwendung - z. T. deutliche Unterschiede. Aus diesem Grund wird eine HV-Lösung für Datenbanksysteme in den meisten Fällen eine individuell angepasste Lösung sein.

Die im Beitrag „Prinzipien der Verfügbarkeit“ beschriebenen Prinzipien werden auch in DBMS durch technische Mechanismen angewandt. Im nächsten Abschnitt werden die Lösungen zur Realisierung von Hochverfügbarkeit auf unterschiedlichen Ebenen vorgestellt.

2 DB-Architekturen

Datenbank-Managementsysteme (DBMS) sind in der Regel eigenständige, komplexe IT-Systeme, die an unterschiedlichen Stellen der Architektur Ansätze für einen hochverfügbaren Betrieb bieten. Einen Ansatz für die Modellierung der Datenbankarchitektur bietet das ANSI-SPARC 3-Schema-Konzept. Dieses beschreibt die grundlegende Trennung verschiedener Beschreibungsebenen für ein Datenbankschema. Dadurch kann eine formalisierte Darstellung der Datenbankarchitektur anhand der Ebenen und Sichten erfolgen (siehe Abbildung 2).

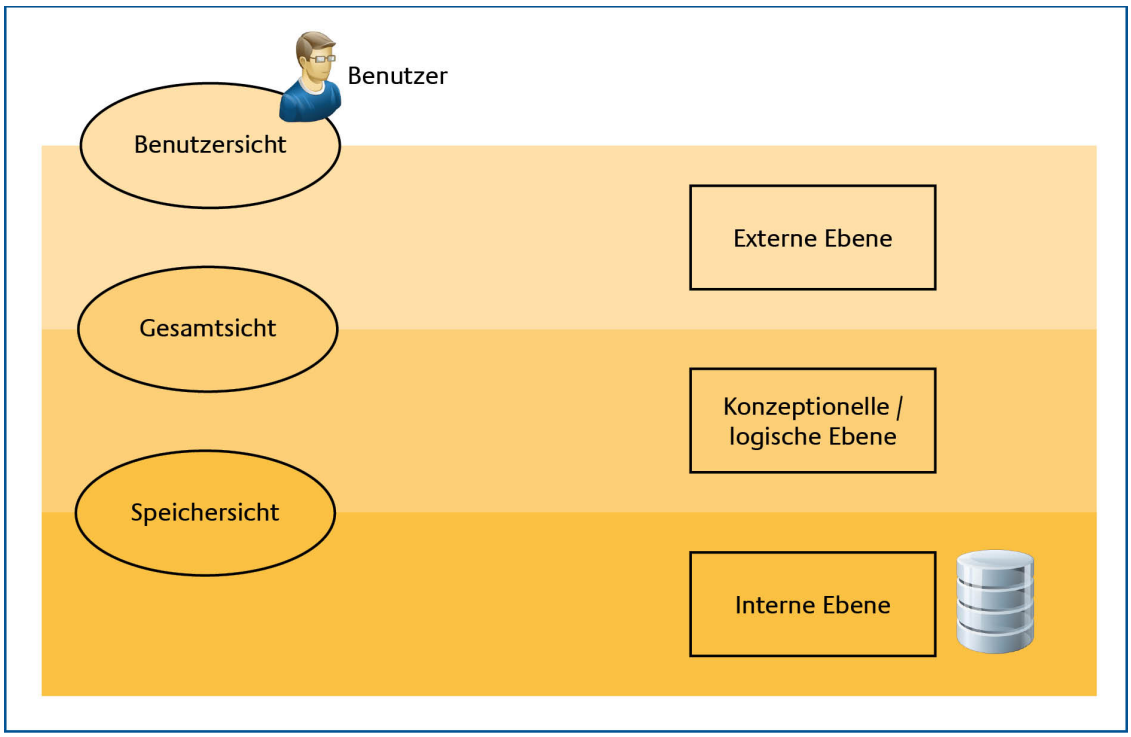


Abbildung 2: ANSI-SPARC 3-Schema-Konzept

Das dargestellte ANSI-SPARC 3 Schema-Konzept wird in den nachfolgenden Absätzen erläutert.

Benutzersicht auf die externe Ebene

Der Zugriff auf die Daten erfolgt typischerweise über eine Benutzerschnittstelle (Frontend) oder über eine Anwendung. Dieser Zugriff auf die Datenbank ist in der Regel über ein Netzwerk (LAN) realisiert und wird in den meisten Fällen über eine standardisierte Datenbanksprache (etwa SQL), Protokolle (etwa LDAP) und Schnittstellen (etwa ODBC oder JDBC) durchgeführt. Daher sind neben der hochverfügbaren Auslegung der Applikation auf Client- und Server-Seite geeignete Mechanismen auf Netzwerkebene (Lastverteilung, QoS der Netzinfrastruktur) zu implementieren (siehe Beitrag „Netzwerk“ im HV-Kompendium).

Gesamtsicht auf die Funktionalität (konzeptionell/ logische Ebene) der Datenbank

Auf dieser Ebene greifen Replikations- und Synchronisationsmechanismen, die sowohl regelmäßige Replikationen der gesamten Datenbank sowie intelligente Verfahren zur Verteilung einzelner Anfragen auf Transaktionsebene realisieren. Bei der Planung von hochverfügbaren Datenbank-systemen ist es auf dieser Ebene wichtig, das voraussichtliche Nutzungsverhalten der Datenbank zu

analysieren. Systeme, die vorwiegend Daten zum Lesen bereitstellen, lassen sich leichter in Clustern betreiben, als solche, die häufige transaktionsorientierte Schreibvorgänge (siehe Kapitel „Optimierung auf Transaktionssicherheit“) durchführen müssen und bei denen auch die Reihenfolge eine Rolle spielt.

Speichersicht der internen Ebene (Backend)

Die tatsächliche (physikalische) Speicherung der Daten ist auf der internen Ebene realisiert. Lösungsansätze liegen in der Speicherung von lesbaren Textdateien, eigenen binären Speicherformaten in Dateien oder Gerätedateien des Betriebssystems (sogenannte Raw-Devices) bis hin zur Nutzung von weiteren Subsystemen wie NAS- oder SAN-Komponenten (siehe Beitrag „Speichertechnologien“ im HV-Kompendium).

Neben der Darstellung der verschiedenen Beschreibungsebenen veranschaulicht die folgende Grafik (siehe) die Verzahnung der verschiedenen Beschreibungsebenen sowie die Interaktion mit den Datenbankkomponenten.

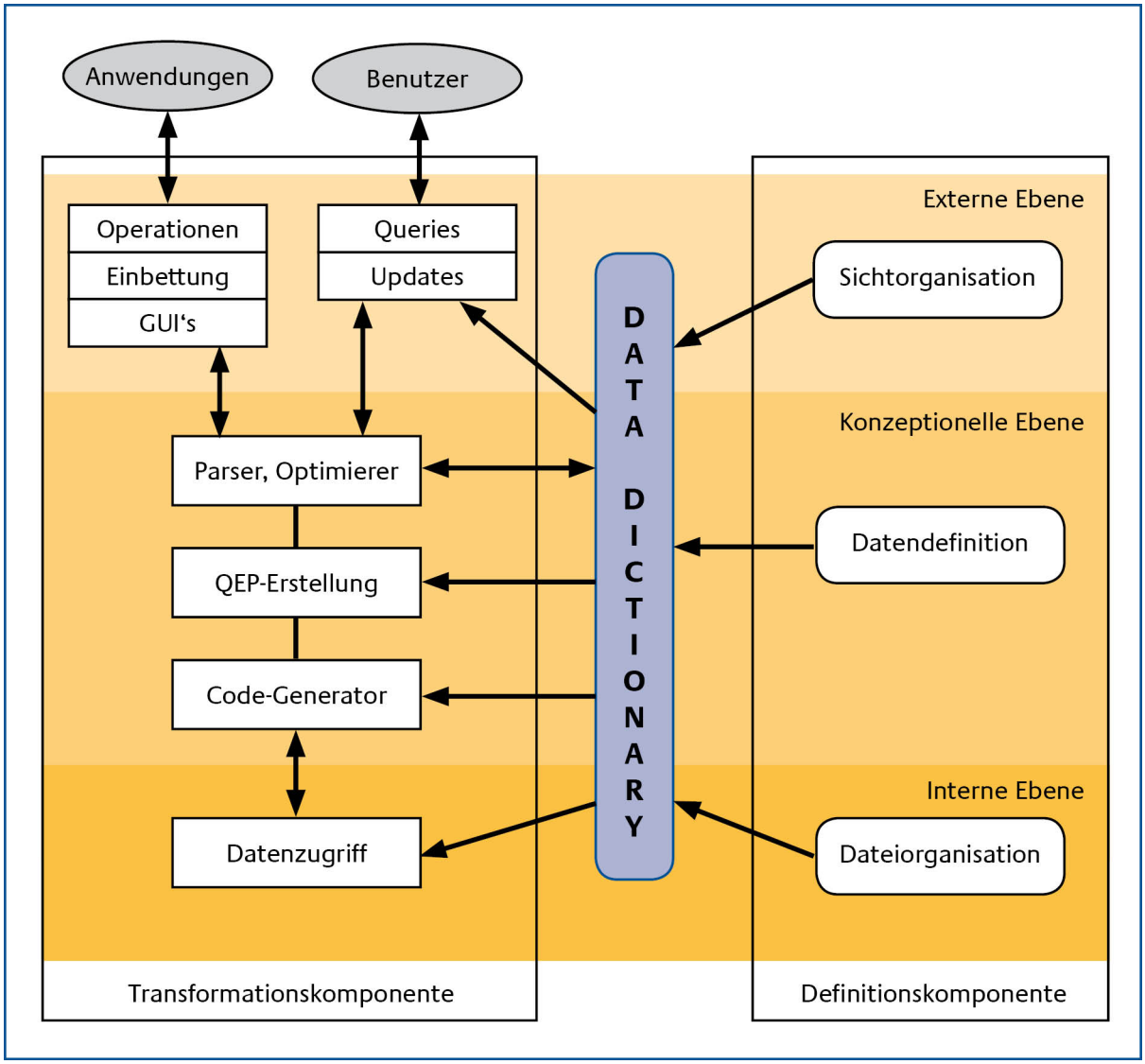


Abbildung 3: Verzahnung der Beschreibungsebenen (nach [Kreißl05])

HV-Aspekte

Eine Verbesserung der Verfügbarkeit kann nur dann erreicht werden, wenn auf verschiedenen Ebenen Technologien zum Einsatz kommen, die jede für sich eine adäquate Verfügbarkeit garantieren und weiterhin über die komplette Kommunikationsstrecke hinweg eine bedarfsgerechte Gesamtverfügbarkeit sicherstellen. Es ist zu prüfen, welche Technologien bereits zum Funktionsumfang der eingesetzten Produkte gehören und welche zusätzlich konzipiert und umgesetzt werden müssen (Individualentwicklung). In den folgenden Abschnitten werden Ansätze auf der konzeptuell/logischen sowie der internen Ebene beschrieben. Lösungsansätze für die externe Ebene haben einen stärkeren Bezug zu den übergeordneten Technologien, die für die gesamte Anwendung wichtig sind, und werden deshalb im Beitrag „Software“ des HV-Kompendiums beschrieben. Bei der Planung und Umsetzung dieser Ansätze muss in jedem Fall auch auf eine geeignete Unterstützung des Managements sowie der Administration der Gesamtarchitektur geachtet werden.

Die unterschiedlichen Sichten auf die DB beschreiben ansatzweise auch unterschiedliche Erwartungen an die Verfügbarkeit einer DB. So sind aus Benutzersicht sowohl die Verfügbarkeit der Daten als auch der DB-Services im Zusammenhang mit der von ihm genutzten Anwendung von Interesse. Die Datenbank muss Daten und Funktionen in einer Qualität zur Verfügung stellen, dass der Benutzer in seinen Arbeitsabläufen nicht gestört wird.

Die eingesetzten Produkte müssen daher neben der Bereitstellung der Sicherheitsmechanismen auch die Handhabbarkeit und Wartbarkeit des Gesamtsystems sicherstellen. Die folgende Auflistung stellt einen Überblick über wesentliche Funktionen dar, die im HV-Umfeld direkt (Out-of-the-Box) oder indirekt (Zusatzwerkzeuge) von den DBMS zur Verfügung gestellt werden:

- Funktionen zur Synchronisation und Replikation des Datenbestandes,
- Unterstützung bei der Einrichtung zusätzlicher redundanter Instanzen in einem Cluster,
- Einrichtung und Betrieb von Standby-Datenbanken,
- Backup- und Recovery-Funktionen,
- Einrichten von Überwachungsfunktionen,
- Einrichtung virtueller Server, IP-Adressen und Verzeichnisse.

Es ist die Aufgabe im Rahmen des DBS-Designs eine DB-Architektur zu entwerfen, die hoch-verfügbar ist und dadurch die bestmögliche Unterstützung der Geschäftsprozesse gewährleistet. Dabei können die genannten Funktionen und Mechanismen hilfreich sein. In welcher Art und Weise sie zur Anwendung kommen ist individuell zu definieren und wird in den nächsten Abschnitten thematisiert.

Die Funktionen zur Replikation und Synchronisation kommen insbesondere in Architekturen zum Einsatz, bei denen mehrere Datenbankserver parallel (Mehrrechner-Datenbanksysteme, Standby-Datenbank) betrieben werden. Neben der Synchronisation/Replikation kann auch eine Verbesserung der Verfügbarkeit durch die Anwendung von Redundanz durch Datenbank-Cluster erzielt werden. Diese soll sicherstellen, dass durch die Erhöhung der Redundanz, in Kombination mit geeigneten (DBMS-) Clustermechanismen, eine hohe Verfügbarkeit gewährleistet wird. Neben dem Bestreben die Verfügbarkeit durch Erhöhung der Anzahl beteiligter Serversysteme zu verbessern, bieten Virtualisierungstechniken weitere Möglichkeiten die Verfügbarkeit zu steigern.

Der Einsatz von Virtualisierungsverfahren kann dabei unterstützen, indem diese zur Minimierung potentieller SPoF beitragen und so die Verfügbarkeit partiell (Virtuelle Verzeichnisse, Virtuelle IP-Adresse) oder grundsätzlich (Servervirtualisierung) erhöhen**.** Neben dem Streben nach einer hohen Verfügbarkeit darf nicht außer acht gelassen werden, dass ein einmal erreichter Verfügbarkeitsstatus lediglich einen Status-Quo darstellt. Es ist daher unerlässlich den Systemzustand permanent zu überwachen (Monitoring und Audit) und für den Schadensfall Backup- und Recovery-Funktionen vorzusehen, die in der Lage sind diesen Status aufrechtzuerhalten oder zumindest schnell wieder herzustellen.

Die hier aufgezeigten Aspekte der Hochverfügbarkeit stellen einen Querschnitt möglicher Funktionen und Mechanismen dar und kommen in unterschiedlichen technischen Ausprägungen zum Einsatz. Sie bilden damit die Grundlage für den Aufbau hochverfügbarer Datenbank-Architekturen, welche in den folgenden Kapiteln beschrieben werden.

2.1 Verteilte DBS

Verteilte Datenbankmanagementsysteme (VDBMS) gestatten eine Anpassung der Systemstruktur an die jeweilige Organisationsstruktur, ohne die Konsistenz der Datenbank zu beeinträchtigen. Zur Darstellung der besonderen Anforderungen an Verteilte DBMS postulierte Date [Da90] zwölf „Regeln“, die die Anforderungen an bestimmte Einzelaspekte Verteilter DBMS darstellen.

- 1. **Lokale Autonomie**
  Jeder Rechner sollte ein Maximum an Kontrolle über die bei ihm gespeicherten Daten haben. Insbesondere sollte der Zugriff auf diese Daten nicht von anderen Rechnern abhängen.
- 2. **Keine Abhängigkeit von zentralen Systemfunktionen**
  Zur Unterstützung einer hohen Knotenautonomie und Verfügbarkeit sollte die Datenbankverarbeitung nicht von zentralen Systemfunktionen abhängen. Solche Komponenten stellen zudem einen potentiellen Leistungsengpass dar.
- 3. **Hohe Verfügbarkeit**
  Die Datenbankverarbeitung sollte trotz Fehler im System (z. B. Rechnerausfall) oder Konfigurationsänderungen (Installation neuer Software oder Hardware) idealerweise nicht unterbrochen werden.
- 4. **Ortstransparenz**
  Die physische Lokalisation von Datenbankobjekten sollte für den Benutzer verborgen bleiben. Der Datenzugriff sollte wie auf lokale Objekte möglich sein.
- 5. **Fragmentierungstransparenz**
  Eine Relation der Datenbank sollte verteilt an mehreren Knoten gespeichert werden können. Die dabei zugrunde liegende (horizontale oder vertikale) Fragmentierung der Relation sollte für den Datenbankbenutzer transparent bleiben.
- 6. **Replikationstransparenz**
  Die replizierte Speicherung von Teilen der Datenbank sollte für den Benutzer unsichtbar bleiben; die Wartung der Redundanz obliegt ausschließlich der DB-Software.
- 7. **Verteilte Anfrageverarbeitung**
  Innerhalb einer DB-Operation (SQL-Anweisung) sollte auf Daten mehrerer Rechner zugegriffen werden können. Zur effizienten Bearbeitung sind durch das Verteilte DBS geeignete Techniken bereitzustellen (Query-Optimierung).

- 8. **Verteilte Transaktionsverwaltung**
  Die Transaktionseigenschaften sind auch bei verteilter Bearbeitung einzuhalten, wozu entsprechende Recovery- und Synchronisationstechniken bereitzustellen sind (verteiltes Commit-Protokoll, Behandlung globaler Deadlocks, u. a.).
- 9. **Hardware-Unabhängigkeit**
  Die DB-Verarbeitung sollte auf verschiedenen Hardware-Plattformen möglich sein. Sämtliche Hardware-Eigenschaften sind für den DB-Benutzer zu verbergen.
- 10. **Betriebssystemunabhängigkeit**
  Die DB-Benutzung sollte unabhängig von den eingesetzten Betriebssystemen sein.
- 11. **Netzwerkunabhängigkeit**
  Die verwendeten Kommunikationsprotokolle und -netzwerke sollten ohne Einfluss auf die DB-Verarbeitung bleiben.
- 12. **Datenbanksystemunabhängigkeit**
  Es sollte möglich sein, unterschiedliche (heterogene) Datenbanksysteme an den verschiedenen Rechnern einzusetzen, solange sie eine einheitliche Benutzerschnittstelle (z. B. eine gemeinsame SQL-Version) anbieten.

Diese 12 „Regeln“ beschreiben Einzelaspekte Verteilter DBMS. Grundsätzlich lässt sich eine Verteilte DBMS dadurch charakterisieren, dass eine logische Datenbank unter mehreren Datenbanksystemen physisch aufgeteilt wird, wobei die einzelnen VDBMS in der Regel auf verschiedenen und üblicherweise geografisch verteilten Rechnern laufen (siehe).

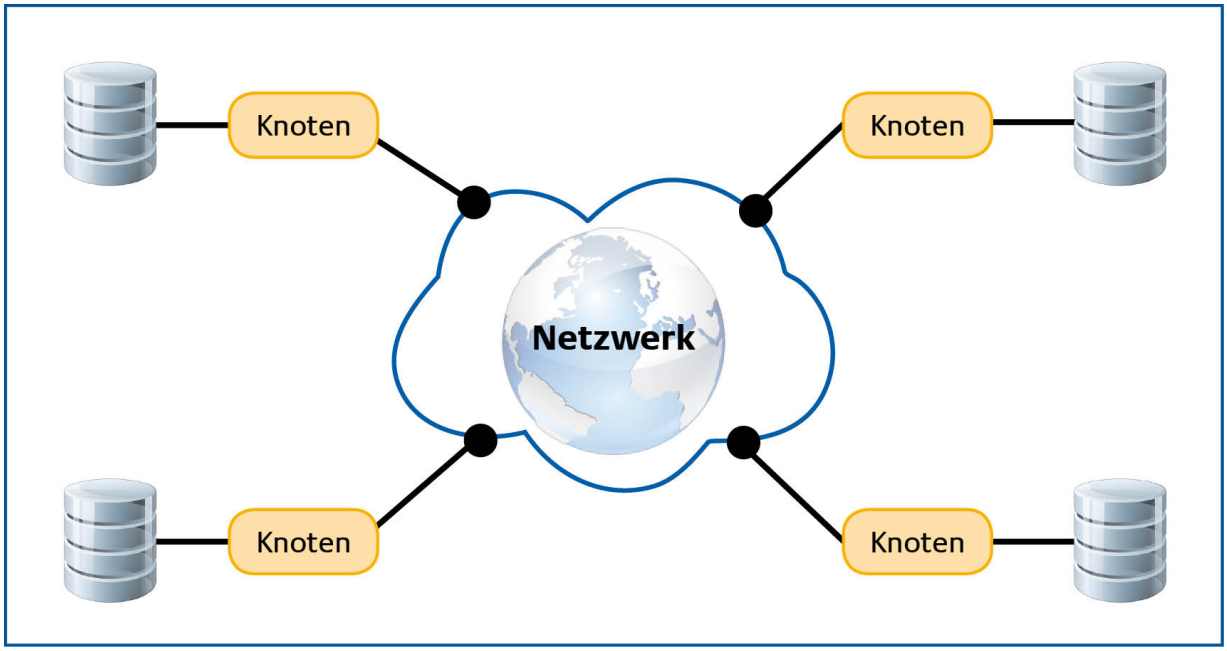


Abbildung 4: Verteiltes DBMS

Die VDBMS der einzelnen Rechner kooperieren dabei eng miteinander, um Datenbankoperationen auf dem verteilten Datenbestand zu bearbeiten. Durch die Kooperation auf Ebene der Datenbanksysteme können gegenüber Anwendungsprogrammen und Benutzern sämtliche Aspekte der Verteilung verborgen werden (Transparenz); für sie ergibt sich keine Änderung in der

Zugriffsschnittstelle gegenüber einem zentralen DBMS. Die Gewährleistung einer solchen Verteilungstransparenz ist die Hauptanforderung an Verteilte Datenbanksysteme. Sie ist wesentlich für eine einfache Benutzbarkeit des verteilten Systems, da trotz der Verteilung sämtliche Funktionen der Datenverwaltung Aufgabe der Datenbank-Software bleiben. Insbesondere bleiben somit Änderungen in der Datenverteilung ohne Auswirkungen auf bestehende Anwendungsprogramme (siehe Abbildung 4).

Neben der Unterstützung dezentraler Organisationsformen bieten Verteilte Datenbanksysteme eine Reihe wesentlicher Vorteile bezüglich Leistungsfähigkeit, Verfügbarkeit und Kosteneffektivität. Ein einzelner Rechner kann leicht zum Systemengpass (SPoF) werden und somit Durchsatz und Antwortzeiten beim Datenbankzugriff beeinträchtigen. Bei Verteilten Datenbanksystemen steht dagegen die Verarbeitungskapazität mehrerer Rechner zur Verfügung; zudem kann die Leistungsfähigkeit durch Erhöhung der Rechneranzahl inkrementell erweitert werden.

In einer zentralisierten Systemarchitektur kann der Ausfall des zentralen DB-Rechners zu einem Ausfall der gesamten Applikation und somit zum Stillstand der Geschäftsprozesse führen. Im verteilten Fall dagegen betrifft ein Rechnerausfall lediglich die von ihm verwaltete Datenmenge. Die DB-Verarbeitung auf den übrigen Rechnern (an anderen Orten) bleibt davon unberührt.

Eine weitere Steigerung der Verfügbarkeit wird erreicht, wenn Teile der Datenbank - unter Kontrolle der DBMS - repliziert gespeichert werden. Damit kann nach einem Rechnerausfall weiterhin auf die gesamte Datenbank zugegriffen werden, wenn die vom Ausfall betroffenen Daten auf einem weiteren Rechner vorliegen. In letztem Fall kann eine ausreichend hohe Verarbeitungskapazität oft nur durch Großrechner (Mainframes) und damit zu hohen Kosten erreicht werden. Verteilte Datenbanksysteme erlauben dagegen die Nutzung mikroprozessorbasierter Rechnerknoten mit typischerweise weit geringeren Kosten pro MIPS als Großrechner.

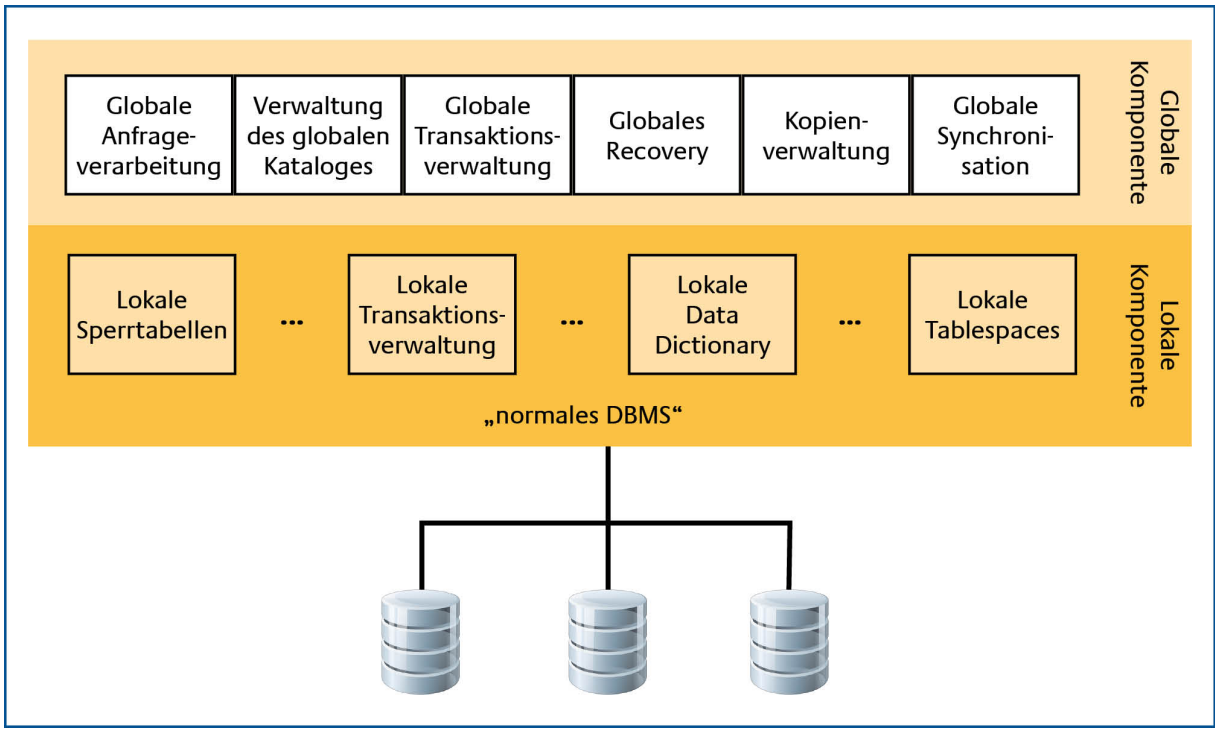


Abbildung 5: Systemarchitektur bei Verteilten DBMS

Unter dem Oberbegriff Verteilte DBS lassen sich die drei folgenden Ausprägungen Verteilter DBS charakterisieren.

Homogen Verteiltes DBS

Bei einem Neuaufbau einer VDBMS lassen sich alle Vorteile einer Verteilten Datenbank durch ein homogen Verteiltes DMBS nutzen. Das heißt, alle vorhandenen DBMS sind vom gleichen Typ und gleicher Ausprägung, sodass keine Integration typfremder DBMS erforderlich ist. Insbesondere können hier die Vorteile wie Ausfallsicherheit und inkrementelle Erweiterbarkeit sehr gut genutzt werden. Ein weiterer Vorteil stellt die Tatsache dar, dass bei einem homogen Verteilten DBMS eine Heterogenität der lokalen Schemata vermieden wird, was wiederum der Leistungsfähigkeit zugute kommt. In der Praxis ist der Aufbau eines homogen Verteilten DBS allerdings nur dann möglich, wenn ein neues DBS aufgebaut wird. Ist bereits ein DBS vorhanden, bieten sich andere Verfahren an.

Homogen integriertes DBS

Da in der Praxis der Ansatz eines homogen Verteilten DBS nicht immer realisierbar ist, kann in diesen Fällen der Ansatz eines homogen integrierten DBS gewählt werden. Hierbei werden autark vorhandene DBMS gleichen Typs und gleicher Ausprägung zu einem VDBMS nachträglich zusammengefasst. In jedem DBMS existieren dabei bereits Daten, die es in einem verteilten System zu konsolidieren gilt. Dabei ist darauf zu achten, dass während des Zusammenfassens immer die Konsistenz der bereits vorhandenen Daten gewährleistet bleibt. Die nachträgliche Integration in ein VDBMS ist allerdings mit einigen Einschränkungen verbunden, da für existierende Anwendungen die alte „Sicht“ weiter verfügbar sein muss.

Heterogen integriertes DBS

Eine weitere Variante stellt das heterogen integrierte DBS dar. Der Haupteinsatzzweck ist das Zusammenführen bereits vorhandener Datenbestände (Legacy-Systeme).

Hierdurch können durch den integrierten Zugriff beispielsweise globale Anfragen aus verschiedenen Datenbanken durchgeführt werden. Aufgrund der Existenz unterschiedlicher Datenmodelle erweist sich die damit einhergehende Heterogenität auf den verschiedenen Ebenen (Datenmodell, Schema) allerdings als problematisch. Bei der Integration von Web-Anwendungen erhält diese Variante aber aktuell eine wichtige Bedeutung.

Anforderungskriterien an VDBS hinsichtlich HV-Anforderungen

Mit der Verwendung eines Verteilten Datenbanksystems ist auch immer die Festlegung der physischen Aufteilung des Datenbestandes verbunden. Diese Festlegung ist für die gesamte Leistungsfähigkeit des Systems von großer Bedeutung. Die physische Aufteilung hat beispielsweise erhebliche Auswirkung auf den Kommunikations- und Organisationsaufwand des DBS. Weiterhin ergeben sich durch die Datenverteilung Rückwirkungen auf die Lastverteilung, da die einem Rechner zugeordneten Daten die von ihm zu verarbeitenden Operationen bestimmen. Durch die Möglichkeit, Teile der Datenbank repliziert zu speichern, wirkt sich die Datenverteilung auch positiv auf die Verfügbarkeit aus.

Im Kern lassen sich die Anforderungen in zwei Aspekte aufteilen: Fragmentierung und Allokation. Im Rahmen der Fragmentierung werden zunächst die Einheiten der Datenverteilung (Fragmente) festgelegt. Anschließend wird über die Allokation (Ortszuweisung) bestimmt, welchem Rechner jedes Fragment zugeordnet wird, wobei eine replizierte Allokation von Fragmenten möglich ist.

Fragmentierung

Im einfachsten Fall stellen ganze Relationen die kleinsten Verteilungsgranulate dar. Dadurch entfällt zum einen die Notwendigkeit einer expliziten Fragmentierung und zum anderen vereinfacht sich durch die geringere Anzahl von Verteilungseinheiten, auch das Allokationsproblem. Darüber hinaus sind damit auch geringere Kommunikationskosten verbunden, da Operationen auf einer Relation stets an einem Rechner ausführbar sind.

Die aufgezeigten Gründe, die für eine Datenverteilung auf Relationenebene sprechen, sind aber gegenüber den folgenden Vorteilen (nach [Rahm94]) einer feineren Fragmentierung abzuwägen:

- Lastverteilung
  Die Allokation vollständiger Relationen ist oft nicht flexibel genug, um eine effektive Nutzung aller Rechner zu erlauben. Somit wird es in der Praxis nur sehr schwer möglich sein die Relationen so unter den Rechnern aufzuteilen, dass eine gleichmäßige Lastverteilung erreicht wird.
- Nutzung von Lokalität
  Durch die Nutzug von Fragmentierung und partitionierter Allokation können Vorteile, z. B. hinsichtlich der Kommunikationsvorgänge, entstehen. Wenn ein hoher Grad von Zugriffslokalität vorliegt, lässt sich dadurch z. B. ein Grossteil der Kommunikationsverbindungen reduzieren.
- Reduzierung des Verarbeitungsumfangs
  Wenn Operationen auf ein Fragment beschränkt werden können, ergibt sich dadurch eine Verringerung der zu verarbeitenden Datenmenge. Damit ist wiederum eine effizientere Bearbeitung gegenüber der Ausführung auf der gesamten Relation gegeben.
- Unterstützung von Parallelverarbeitung
  Durch die Fragmentierung können Operationen auf einer Relation in Teiloperationen auf Fragmenten unterschiedlicher Rechner zerlegt werden. Dadurch wird es möglich, dass diese zur Verkürzung der Bearbeitungszeit parallel ausgeführt werden können.

Die Zerlegung einer globalen Relation in Fragmente kann grundsätzlich auf zwei Arten erfolgen. Zum einen ist eine horizontale (zeilenweise) und zum anderen eine vertikale (spaltenweise) Fragmentierung möglich. Um die Korrektheit der Fragmentierung sicherzustellen, sind die folgenden drei Regeln (nach [Rahm94]) zu beachten:

- **Vollständigkeit**
  Jedes Datenelement (Tupel, Attributwert) der globalen Relation muss in wenigstens einem Fragment enthalten sein.
- **Rekonstruierbarkeit**
  Die Zerlegung muss verlustfrei sein, sodass die globale Relation aus den einzelnen Fragmenten wieder vollständig rekonstruiert werden kann.
- **Disjunktheit**
  Fragmente sollten möglichst disjunkt sein, da ihre Replikation im Rahmen der Allokation festgelegt wird.

Allokation und Replikation

Nach Festlegung der Fragmentierung für globale Relationen besteht der zweite Schritt bei der Datenverteilung in der Allokation der Fragmente zu Knoten. Falls keines der Fragmente mehrfach allokiert wird, erhält man eine partitionierte Datenbank, anderenfalls eine replizierte Datenbank.

Gerade vor den Anforderungen im HV-Bereich verbindet man mit der replizierten Datenhaltung ein wesentliches Ziel: die Steigerung der Verfügbarkeit. Denn im Gegensatz zur partitionierten Datenallokation kann auf replizierte Objekte auch nach dem Ausfall eines Speicherknotens zugegriffen werden. Es ergeben sich allerdings, neben dem Aspekt der Verfügbarkeit auch Ansatzpunkte zur Verbesserung der Performance. So ergeben sich durch die Replikation für Anwendungen, die eine hohe Lese-Performance erfordern, ergeben sich so Optimierungsmöglichkeiten. Denn wenn mehrere Knoten ein bestimmtes Fragment führen, kann ein kostengünstiger Ausführungsplan erstellt werden. Besonders gilt dieser Aspekt für ein repliziertes Objekt, auf das an mehreren Knoten lokal referenziert werden kann; dadurch können Kommunikationsvorgänge gegenüber einer partitionierten Datenbankallokation eingespart werden. Weiterhin erhöhen sich die Möglichkeiten zur Lastverteilung, da bestimmte Datenzugriffe von mehreren Rechnern ausführbar sind.

Aufgrund der geforderten Replikationstransparenz ist die Wartung der Replikation alleinige Aufgabe des Verteilten DBS. Für den Benutzer bleibt die Existenz der Replikate vollständig verborgen. Ebenso unsichtbar bleibt die Lokation einzelner Fragmente (Ortstransparenz).

2.1.1 Betrachtung der Realisierungsalternativen

Verteilte DBS stellen einen zeitgemäßen Ansatz dar, um den aktuellen Anforderungen von Geschäfts- und IT-Prozessen zu entsprechen. Sie verfügen neben Redundanz und Transparenz auch über Eigenschaften, die sie im Rahmen des Einsatzes im HV-Umfeld prädestinieren. Hierzu zählen insbesondere die folgenden Merkmale:

- Unanfälligkeit gegen ggf. auftretende Fehler wie z. B. dem Ausfall eines Rechnerknotens (Vermeidung von Single Point of Failure)
- Möglichkeiten einer Konfigurationsänderung während des laufenden Betriebs (z. B. neue Software oder Hardware)

Der Einsatz von Verteilten DBS ist jedoch immer im Gesamtkontext zu betrachten, denn neben den bereits aufgezeigten Charakteristika haben die folgenden Faktoren einen erheblichen Einfluss auf die Performance und die Leistungsfähigkeit von Verteilten DBMS:

- Aufwand für Interprozessorkommunikation,
- Möglichkeiten der parallelen Verarbeitung,
- Lastverteilung,
- Cache-Kohärenzkontrolle,
- Synchronisation und
- Kommunikationsaufwand zwischen den einzelnen Knoten.

In der Regel verkürzt ein niedriger Kommunikationsaufwand die Antwortzeiten und erhöht den Durchsatz bei Anfragen und Transaktionen. Eine hohe Parallelität auf der Hardware- und/oder Softwareebene kann Anfragen bzw. Transaktionen deutlich beschleunigen. Eine effiziente Lastverteilung sorgt für eine gleichmäßige Prozessorauslastung und ermöglicht damit eine Beschleunigung der Beantwortung von Anfragen. Ein zu hoher Aufwand für die Cache-

Kohärenzkontrolle und aufwendige Synchronisation erfordern meistens zusätzlichen Kommunikations- und Verwaltungsaufwand und führen zur Verschlechterung der Antwortzeiten sowie der Minimierung des Durchsatzes.

So sind im Kontext der individuellen Anforderungen die verschiedenen Faktoren zu gewichten, um ein optimales Gesamtergebnis zu erreichen.

Neben den hier aufgezeigten Entscheidungshilfen ist allerdings zu beachten, dass für verteilte Systeme immer ein charakteristisches Kernproblem existiert: der Mangel an globalem (zentralisiertem) Wissen.

2.2 Mehrrechner-Datenbanksysteme

Unter der Bezeichnung "Mehrrechner-Datenbanksysteme" werden Architekturen zusammengefasst, bei denen mehrere Prozessoren oder DBMS-Instanzen an der Verarbeitung von DB-Operationen beteiligt sind. Um eine „Isolierung“ der Prozessoren oder DBMS auszuschließen, ist eine Kooperation der Prozessoren bzw. DBMS bezüglich der DB-Verarbeitung notwendig. Auf der Grundlage dieser Definition lassen sich die folgenden Realisierungsvarianten von Mehrrechner-Datenbanksystemen ableiten:

- Externspeicheranbindung (gemeinsam vs. partitioniert),
- Räumliche Verteilung (lokal vs. ortsverteilt),
- Rechnerkopplung (eng, nahe, lose),
- Integrierte vs. föderative Mehrrechner-DBS und
- Homogene vs. heterogene DBMS.

Eine Klassifikation von Mehrrechner-DBS (siehe Abbildung 6) ergibt sich durch die Verwendung der drei Kriterien Externspeicherzuordnung, räumliche Verteilung sowie Rechnerkopplung. Nachfolgend werden diese drei Kriterien näher spezifiziert und bzgl. ihrer HV-Relevanz untersucht.

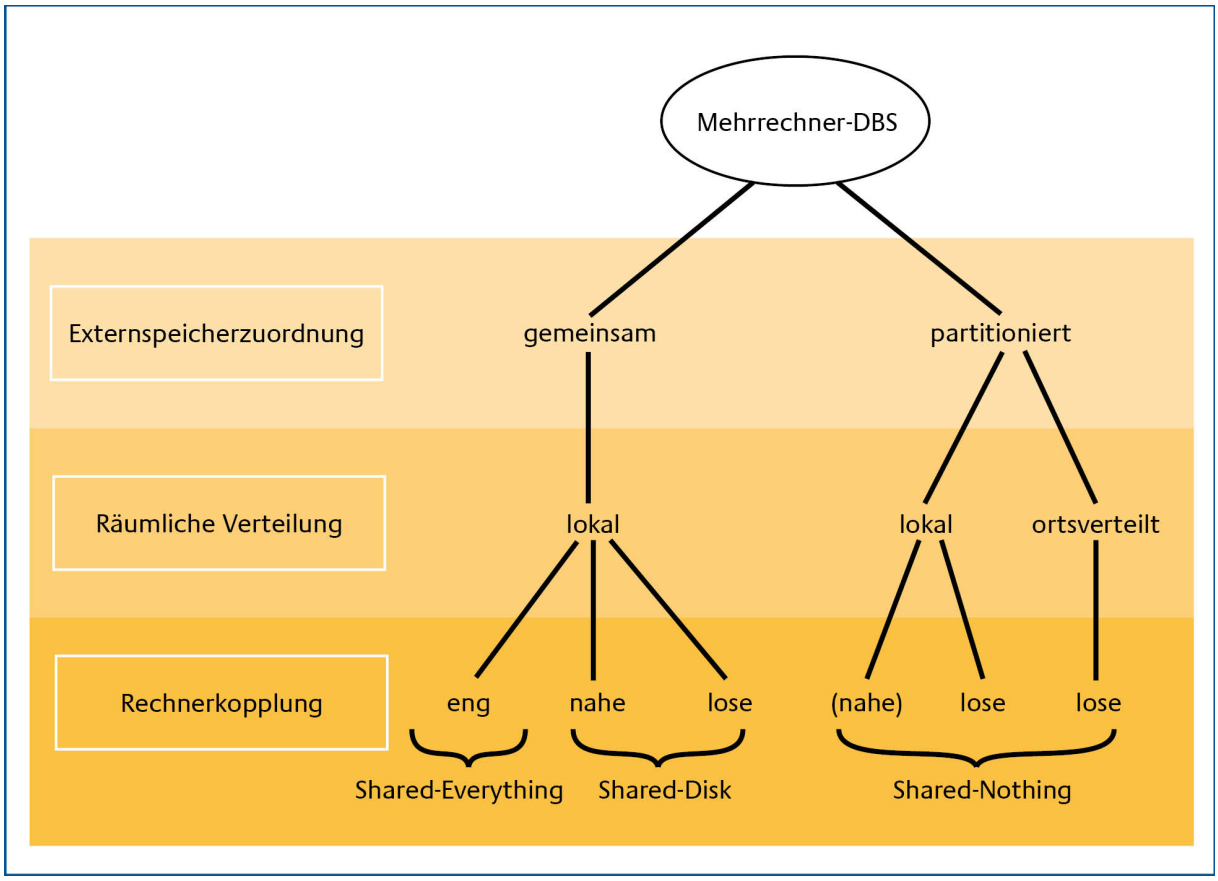


Abbildung 6: Realisierungsvarianten bei Mehrrechner-DBS (nach [Rahm02])

2.2.1 Externspeicheranbindung

Bei der Realisierung der Externspeicheranbindung lassen sich die beiden Varianten, *partitionierter* und *gemeinsamer* Zugriff, unterscheiden.

Beim *partitionierten Zugriff* erfolgt eine Partitionierung der Externspeicher unter den Prozessoren bzw. Rechnern. Dabei erfolgt eine Partitionierung der Externspeicher derart, dass jedem Rechner eine Partition / Laufwerk sowie die darauf gespeicherten Daten zugeordnet werden. Der direkte Zugriff auf diese Partition oder deren Daten ist nur für den entsprechenden Rechner möglich. Ist ein Zugriff auf eine andere Partition erforderlich, so erfordert dies die Kommunikation mit dem Eigner der jeweiligen Partition im Rahmen der Transaktionsverarbeitung. Diese Restriktionen erfordern eine verteilte Ausführung der DB-Operationen bzw. Transaktionen. Ein wesentlicher Aspekt des partitioniertem Zugriffs stellt die Tatsache dar, dass eine Partitionierung der Externspeicher nicht zwingend mit einer Partitionierung der Datenbank einhergeht. So bietet diese Variante die Möglichkeit, die Datenbank mittels partitioniertem Zugriffs ganz oder teilweise repliziert auf mehreren Rechnern zu speichern. Dieser Aspekt bietet zum einen die Möglichkeit der Optimierung von Kommunikationsvorgängen zum anderen kann dies auch dazu genutzt werden, die Fehlertoleranz und damit die Verfügbarkeit zu erhöhen.

Im Gegensatz zum *partitionierten Zugriff* hat beim *gemeinsamen Zugriff* jeder Prozessor direkten Zugriff auf die Externspeicher und damit auf die gesamte Datenbank. Dieser Umstand führt dazu, dass beim *gemeinsamen Zugriff* kein „Overhead“ im Rahmen der Transaktionsverarbeitung entsteht, da die Kommunikation zwischen den „Eignern“ der Externspeicher auf Synchronisationsmaßnahmen reduziert wird. Während im Bereich des *partitionierten Zugriffs* eine

Shared-Nothing-Architektur vorliegt, liegt bei der Variante des *gemeinsamen Zugriffs eine* Shared-Disk- sowie Shared-Everything-Architektur vor.

2.2.2 Räumliche Anordnung

Bei der Klassifikation der räumlichen Anordnung kann zwischen *lokaler* und *ortsverteilter* Rechneranordnung unterschieden werden. Grundsätzlich ermöglichen lokal verteilte Systeme eine wesentlich effizientere Interprozessor-Kommunikation und sind im Allgemeinen robuster gegenüber Fehlern im Kommunikationssystem (z. B. Netzwerkfehler). Diese Eigenschaft ist vor allem für die parallele Datenverarbeitung von Bedeutung, da eine Parallelisierung immer auch einen Mehraufwand an Kommunikation bedingt. Neben diesen Aspekten spielen auch Faktoren wie eine dynamische Verteilung der Transaktionslast und Aspekte der Administration eine nicht zu unterschätzende Rolle. Eine dynamische Verteilung der Transaktionslast ist bei lokal verteilten Systemen durch eine größere Aktualität eher gegeben als bei einer ortsverteilten Rechneranordnung. Zudem ergeben sich Vorteile hinsichtlich der Administration, da nicht in jedem Knoten eine eigene Systemverwaltung erforderlich ist.

Den Vorteilen der lokalen Rechneranordnung stehen aber teilweise die Anforderungen heutiger Geschäftsprozesse und Unternehmensstrukturen entgegen. Denn nur mit ortsverteilten Mehrrechner-DBS kann eine Anpassung an (weltweit) verteilte Organisationsstrukturen erfolgen. Besondere Anforderungen im HV-Umfeld, z. B. Verfügbarkeit auch im K-Fall[1], lassen sich wesentlich besser durch ortsverteilte Konfigurationen abbilden. Diese sind wesentlich robuster gegen den Ausfall z. B. eines kompletten Rechenzentrums.

2.2.3 Rechnerkopplung

Neben den Ansätzen der Räumlichen Rechneranordung gibt es auch den Ansatz mehrere Prozessoren (Rechnerkopplung) miteinander zu verbinden. Dabei lassen sich zwei Arten der Rechnerkopplung unterscheiden, die *enge* und die *lose* Rechnerkopplung.

Bei der *engen Kopplung* (siehe) teilen sich die Prozessoren einen gemeinsamen Hauptspeicher. Dabei liegt die Software (Betriebssystem, DBMS) nur in einer Kopie vor. Die Nutzung solcher Systeme zur DB-Verarbeitung bezeichnet man als *Mehrprozessor-DBS.* Sie werden auch, in Anlehnung der gemeinsamen Ressourcennutzung, als *"Shared-Memory"*- bzw. "*Shared-Everything"*-Systeme bezeichnet. Die *enge Kopplung* gestattet eine effiziente Kooperation zwischen den Prozessoren über einen gemeinsamen Hauptspeicher. Zur Erreichung einer effektiven Lastverteilung ist es aber erforderlich, dass das zugrunde liegende Betriebssystem diese Anforderung unterstützt. Die Nutzung eines gemeinsamen Hauptspeichers bietet allerdings nur eine geringe Fehlerisolation, weshalb bei dieser Art der Kopplung Verfügbarkeitsprobleme entstehen können. Neben potentiellen Verfügbarkeitsproblemen ist bei der *engen Kopplung* auch die Skalierbarkeit mit wachsender Prozessoranzahl stark begrenzt. Mit wachsender Prozessoranzahl steigt auch die Affinität bezüglich potentieller Kohärenz- oder Invalidierungsprobleme. Diese sind dadurch bedingt, dass bei der Verwendung von Prozessor-Caches die Speicherinhalte in den privaten Cache-Speichern der Prozessoren gehalten werden. Dies wiederum führt dazu, dass Änderungen in einem der Caches zu veralteten Daten in den anderen Caches führen. Die erforderliche Kohärenzkontrolle führt jedoch in der Regel zu vermehrten Hauptspeicherzugriffen und Kommunikationsvorgängen, wodurch die Erweiterbarkeit wiederum stark beeinträchtigt werden kann. Schließlich wird die Erweiterbarkeit zusätzlich durch die für den Zugriff auf gemeinsame Hauptspeicher-Datenstrukturen notwendige Synchronisation zwischen Prozessen eingeschränkt, da die Konfliktwahrscheinlichkeit mit der Anzahl zugreifender Prozesse/ Prozessoren zunimmt.

1 Katastrophen-Fall (z. B. Erdbeben)

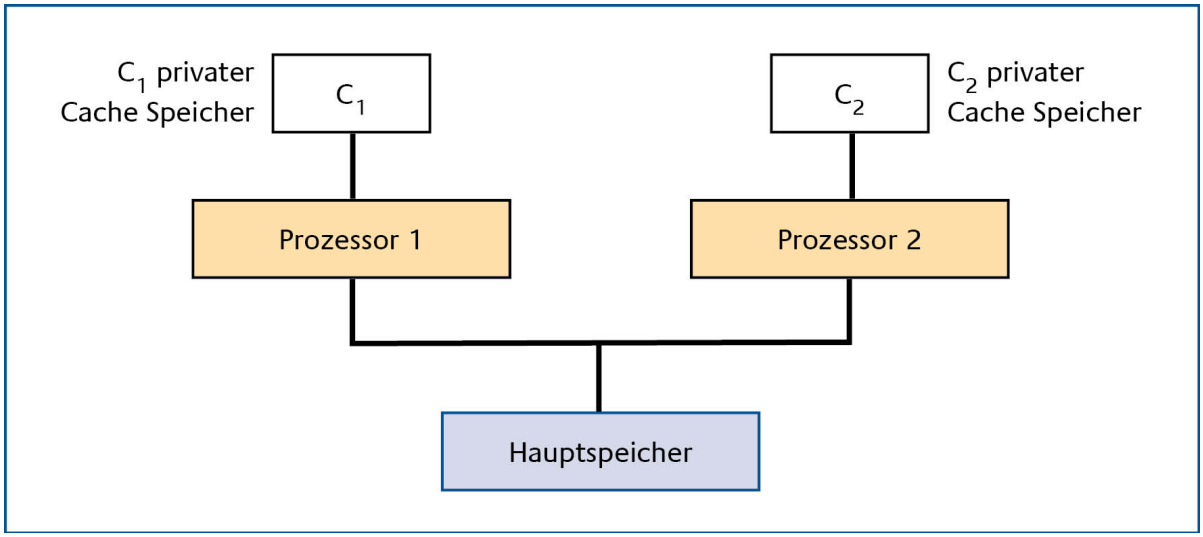


Abbildung 7: Enge Kopplung (nach [Härder02])

Bei der *losen Rechnerkopplung* (siehe Abbildung 8) kooperieren mehrere unabhängige Rechner, die jeweils einen eigenen Hauptspeicher sowie private Software-Kopien (DBMS, Betriebssystem, Anwendungen) besitzen. Durch die Nutzung eines eigenen Hauptspeichers ist bei der losen Rechnerkopplung eine weit bessere Fehlerisolation gegeben als bei *enger Kopplung*. Weiterhin ist durch den Verzicht eines gemeinsamen Hauptspeichers eine bessere Erweiterbarkeit gegeben. Die Skalierbarkeit in Bezug auf die Gesamt-CPU-Kapazität ist bei der losen Rechnerkopplung ungleich besser als bei der engen Rechnerkopplung, denn hier kann jeder Rechnerknoten wiederum ein Multiprozessor sein. Der Hauptnachteil der *losen Kopplung* ist die aufwendige Kommunikation über ein Verbindungsnetzwerk. Dies kann auch der Grund für Kommunikationsverzögerungen sein, die dadurch bedingt sind, dass ein synchrones Warten auf eine Antwortnachricht nicht gegeben ist. Für lose gekoppelte Mehrrechner-DBS ist auf jedem Rechner eine lokale Kopie des DBMS vorhanden. Diese kommunizieren intensiv miteinander um eine koordinierte DB-Verarbeitung zu gewährleisten. Dieses Merkmal macht eine effektive Lastverteilung wesentlich schwieriger als bei einer engen Kopplung und kann auch zu Überlastsituationen führen.

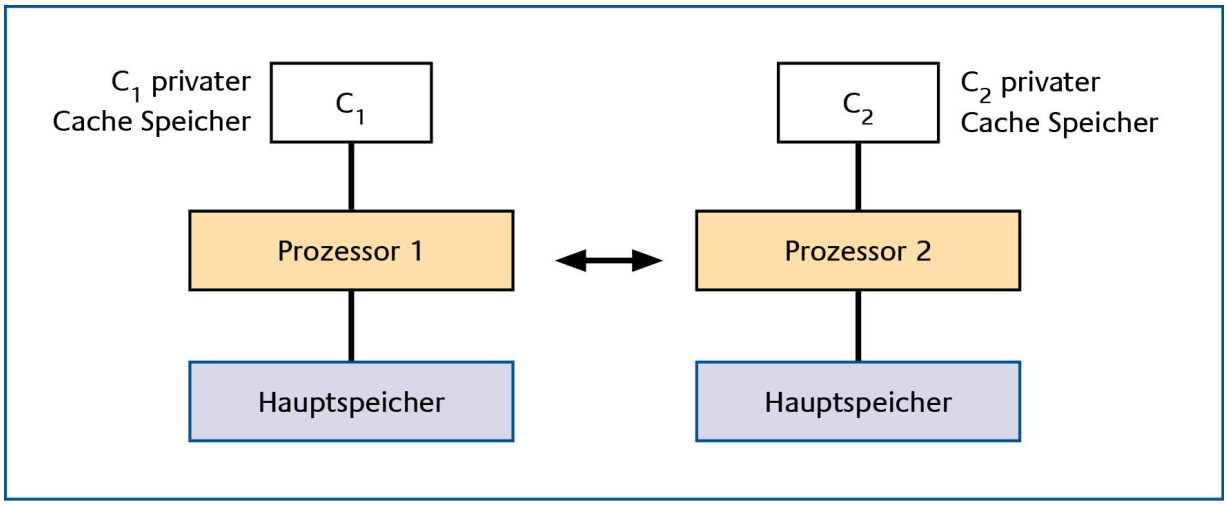


Abbildung 8: Lose Kopplung (nach [Härder02])

Shared-Everything

Eine Variante einer Verteilten DBS stellt die „Shared-Everything“-Architektur (siehe Abbildung 9) dar. Hierunter wird die Nutzung eines gemeinsamen Hauptspeichers und gemeinsamer Platten verstanden, wobei die DB-Verarbeitung durch ein DBMS auf einem Multiprozessor erfolgt.

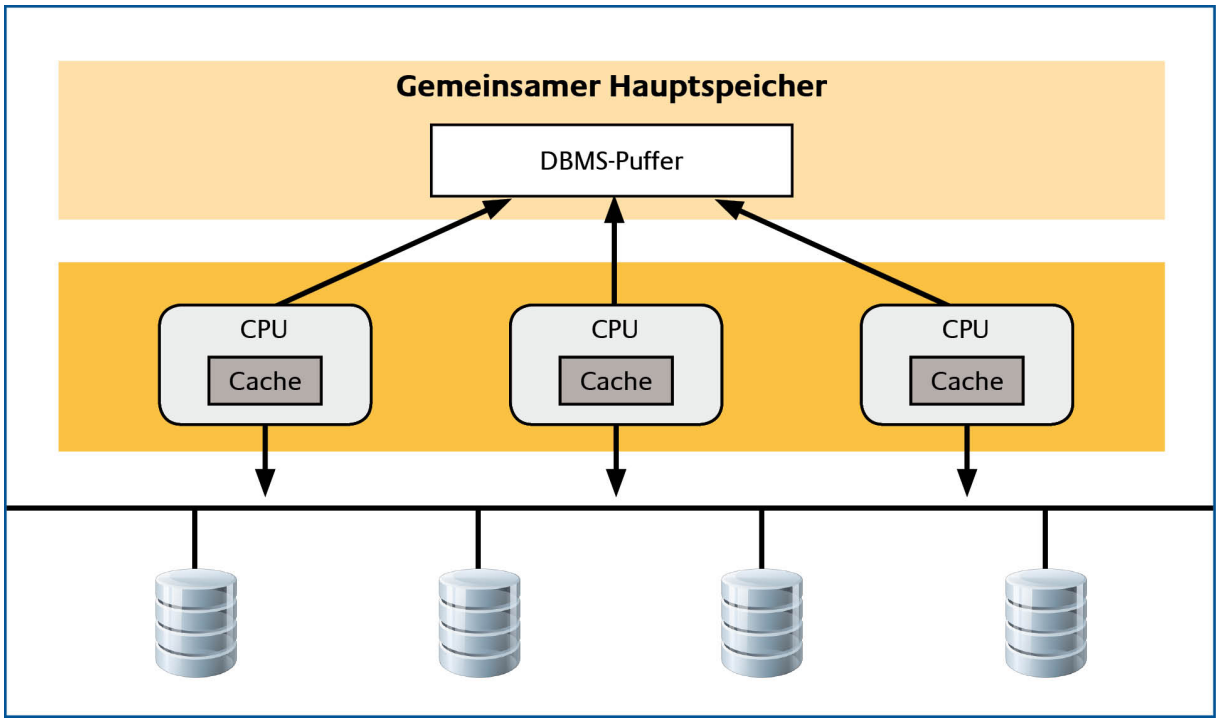


Abbildung 9: Klasse Shared-Everything (nach [Schallehn06])

Von Vorteil ist die gute Skalierbarkeit über die Anzahl der Prozessoren. Allerdings geht damit einher, dass sich mit zunehmender Zahl von Prozessoren die Konflikte und Engpässe (z. B. durch den gemeinsamen Zugriff auf Platten und Hauptspeicher, Cache-Koheränzproblemen) erhöhen. Dies macht eine aufwendige Synchronisation der beteiligten Prozessoren notwendig. Im Hinblick auf die Verfügbarkeitsanforderungen bietet der Shared-Everything-Ansatz wenig bis keinen Nutzen, denn bei Ausfall des gemeinsam genutzten Hauptspeichers sind alle Konten davon betroffen. Dieser Ansatz bietet sich in der Regel an, wenn eine gute Skalierbarkeit mit einem entsprechenden Performancegewinn gewünscht wird.

Shared-Disk

Bei einer Shared-Disk-Architektur (siehe Abbildung 10) wirken komplette Rechner bei der DB-Verteilung zusammen. Jeder einzelne Rechner darf mehrere CPUs aufweisen.

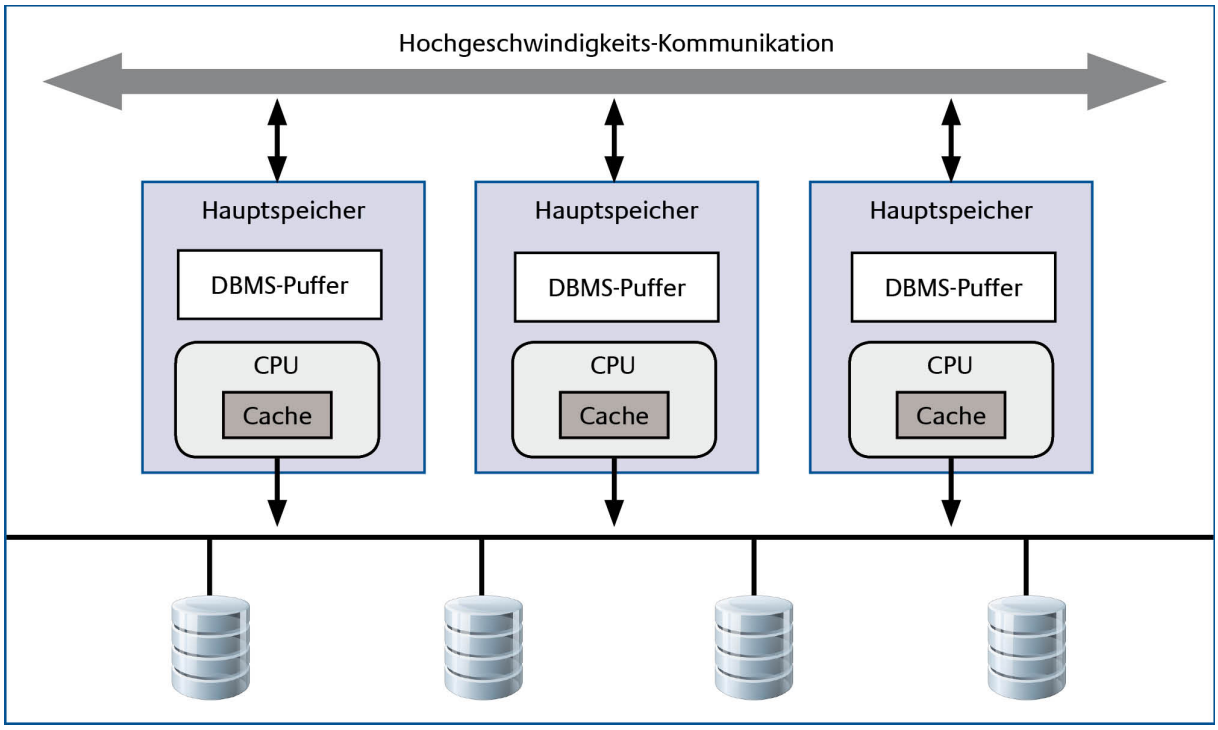


Abbildung 10: Klasse Shared-Disk (nach [Schallehn06])

Eine Skalierbarkeit erfolgt nun auf zwei Ebenen (Ausbau einzelner Rechner und Ausbau des Rechner-Clusters). Aus dieser Architektur ergeben sich unter Betrachtung von Verfügbarkeitsaspekten insbesondere Vorteile beim Ausfall einzelner Rechner-Cluster. Das Gesamtsystem kann auch in diesem Fall weiter betrieben werden, allerdings nur mit eingeschränkter Performance. So ist diese Architektur nicht nur fehlertolerant gegenüber dem Ausfall einzelner Rechner, sondern erlaubt z. B. auch die Wartung einzelner Rechner bei gleichzeitiger Verfügbarkeit des Gesamtsystems. Neben diesen Verfügbarkeitsaspekten bietet diese Architektur auch Vorteile hinsichtlich der Erweiterung des Gesamtsystems. Das Hinzufügen weiterer Rechner zum Cluster ist in der Regel im laufenden Betrieb möglich. Werden Änderungen an der Clusterverteilung durchgeführt, so können diese auch für Verbesserungen in der Lastverteilung eingesetzt werden, um einen optimalen Betrieb zu gewährleisten.

Der Shared-Disk-Ansatz ist eine vielversprechende Möglichkeit, eine hohe Verfügbarkeit bei gleichzeitiger Lastverteilung zu erreichen. Allerdings stellt dieser Ansatz hohe Anforderungen an die Lastverteilung über *n* Rechner im Gesamtsystem. Hier lässt sich durch Verfahren wie Affinity-Based-Routing (separate Systempuffer, Erteilung von Lokalitätsverhalten) verhindern, dass einzelne Rechner zum Engpass werden.

Daneben bedingt die hohe Datenbanklast, dass hohe Anforderungen an eine Konsistenzhaltung der Datenbanksystempuffer gestellt werden. So sind im Fall einer Shared-Disk-Architektur mehrere Sperrtabellen (siehe auch Abschnitt „Verklemmungen (Log-Analyse)“) vorhanden, so dass auch Probleme durch Sperrproblematik (Deadlocks) auftreten können. Shared-Nothing

Bei einer Shared-Nothing-Architektur (siehe Abbildung 11) erfolgt die DB-Verarbeitung durch mehrere, im Allgemeinen lose gekoppelte Rechner, auf denen jeweils ein DBMS abläuft.

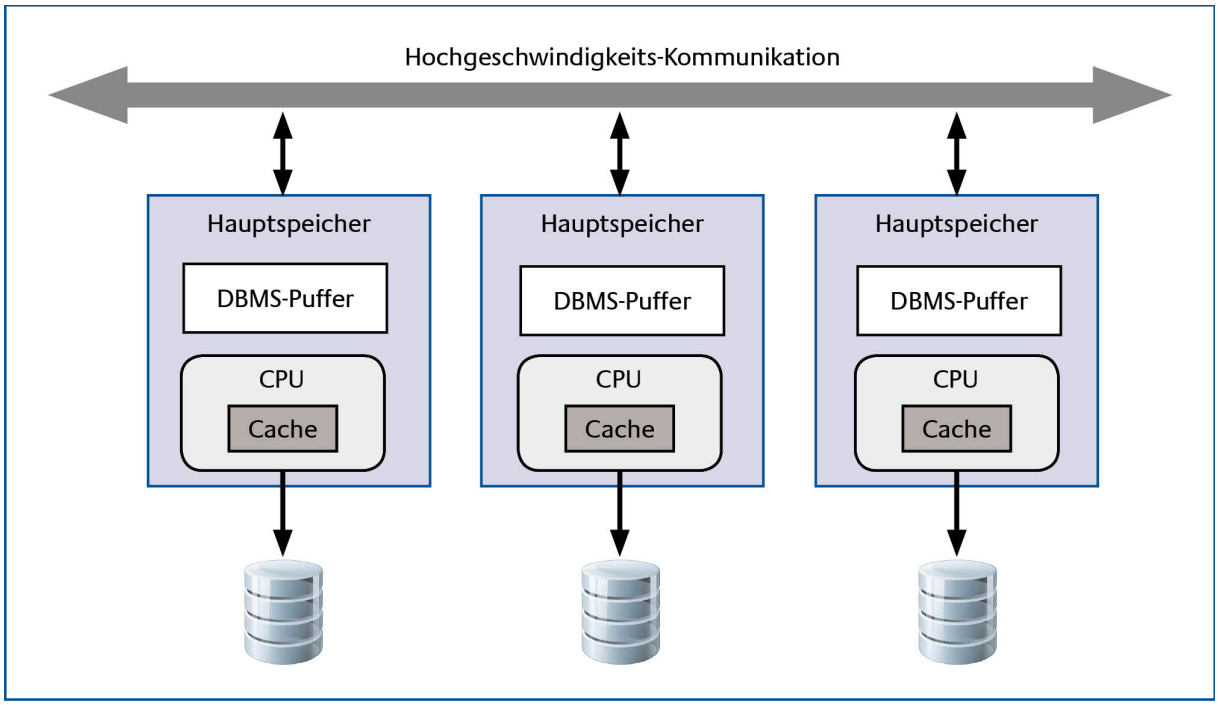


Abbildung 11: Klasse Shared-Nothing (nach [Schallehn06])

Die Externspeicher sind unter den Rechnern partitioniert, sodass jedes DBMS nur auf Daten der lokalen Partition direkt zugreifen kann. Die Rechner können lokal oder geografisch verteilt angeordnet sein. Multiprozessoren sind als Rechnerknoten möglich, das heißt, die lokalen DBMS können vom Typ „Shared-Everything“ sein.

2.2.4 Integrierte vs. föderierte Mehrrechner-DBS

Eine weitergehende Klassifizierung ergibt sich durch die Unterscheidung zwischen integrierten und föderativen sowie zwischen homogenen und heterogenen Mehrrechner-DBS. Diese Merkmale sind nur für Architekturen mit mehreren DBMS relevant, also nicht für Shared-Everything-Architekturen.

Integrierte Mehrrechner-DBS sind dadurch gekennzeichnet, dass sich alle Rechner (DBMS) eine gemeinsame Datenbank teilen, deren Aufbau durch ein einziges konzeptionelles (logisches) Schema beschrieben ist. Durch die Nutzung eines gemeinsamen Datenbankschemas kann den Transaktionsprogrammen volle Verteilungstransparenz geboten werden. Sie sind dadurch in der Lage, über das lokale DBMS wie im zentralisierten Fall auf die Datenbank zugreifen zu können. Diesem Vorteil steht eine starke Einschränkung der Autonomie der einzelnen Rechner/DBMS entgegen. Denn das Durchführen von Änderungen, z. B. Schemaänderungen, aber auch die Vergabe von Zugriffsrechten müssen global koordiniert werden. Ebenfalls wird vorausgesetzt, dass jede DB-Operation auf allen Rechnern gleichermaßen gestartet werden kann und dass potentiell alle DBMS bei der Abarbeitung einer DB-Operation kooperieren. Dies erfordert auf allen Rechnern identische (homogene) DBMS.

Verteilte Datenbanksysteme sind ein Beispiel solch integrierter Mehrrechner-Datenbanksysteme.

Im Vergleich zu den Integrierten Mehrrechner-DBS streben *Föderative Mehrrechner-DBS* nach größerer Knotenautonomie. Dabei können die beteiligten DBMS entweder homogen oder heterogen sein. Die geforderte Unabhängigkeit der einzelnen Datenbanksysteme und Rechner lässt zur Realisierung dieser Systeme nur eine Partitionierung der Externspeicher (Shared-Nothing) zu. Kenn-

zeichnende Eigenschaft der föderativen DBS ist, dass jeder Rechner eine eigene Datenbank verwaltet, die durch ein lokales (privates) konzeptionelles Schema beschrieben ist. Durch die Nutzung einer begrenzten Kooperation der DBMS kann auf bestimmte Daten anderer Rechner zugegriffen werden. Dabei wird die Menge der kooperierenden DBMS als Föderation bezeichnet. Ist eine anwendungsbezogene Definition der Föderation gegeben, kann ein DBS auch an mehreren Föderationen beteiligt sein. Die Voraussetzung dafür ist aber in jedem Fall, dass ein einheitliches Datenmodell der lokalen DBMS zugrunde liegt. Dadurch kann innerhalb einer DB-Operation auf Daten verschiedener Datenbanken zugegriffen werden.

2.3 Parallele Datenbanksysteme

Parallele Datenbanksysteme sind ein Spezialfall der verteilten DBMS. Wie bei den verteilten Datenbanken (siehe Abschnitt „Mehrrechner-Datenbanksysteme“) kooperieren mehrere DBMS miteinander, die jeweils auf einem eigenen Rechner arbeiten. Die Rechner sind jedoch räumlich nahe beieinander (i. A. in einem Rechenzentrum), um eine schnelle Kommunikation zu gewährleisten.

Aufgrund der parallelen Verarbeitung durch zahlreiche Prozessoren lassen sich diese zur Leistungssteigerung nutzen. Dabei muss man nach Art der gewünschten Leistungssteigerung differenzieren: abhängig davon, ob ein größerer Durchsatz an Transaktion erreicht werden soll oder ob die Bearbeitungszeit einer Transaktion verkürzt werden soll. Daher lassen sich die Art und Weise der parallelen DB-Verarbeitung auf die beiden folgenden Arten charakterisieren: Inter- und Intratransaktionsparallelität.

Intertransaktionsparallelität betrifft die parallele Ausführung mehrerer unabhängiger Transaktionen oder DB-Anfragen. Dies führt zu einer Verbesserung des Durchsatzverhaltens. Intra-Transaktionsparallelität betrifft die Parallelverarbeitung innerhalb einer Transaktion. Dabei ist das Hauptziel nicht die Verbesserung des Durchsatzverhaltens, sondern die Verkürzung der Bearbeitungszeit von Transaktionen bzw. eine Verbesserung der Antwortzeiten. Charakteristisch ist dabei eine enge und hochgradig parallele Bearbeitung eines Benutzerauftrags.

Dem theoretisch denkbaren Ansatz, das Antwortzeitverhalten durch die Parallelität immer weiterer, zusätzlicher Prozessoren zu steigern, sind in der Praxis Grenzen gesetzt. So steigt mit zunehmender Prozessoranzahl auch der Aufwand (z. B. Koordinierung der Speicher- und Datenbankzugriffe) der betrieben werden muss, um eine koordinierte Abarbeitung der Prozesse zu gewährleisten. Daher verhält sich die - mit dem Grad der Parallelität erreichte Leistungssteigerung - in der Praxis nicht linear (siehe Abbildung 12). Es kann sogar zu einer Verschlechterung des Antwortzeitverhaltens kommen, wenn der Koordinierungsaufwand ein gewisses Maß überschreitet. Ein Maß um die Leistungsfähigkeit eines Computersystems nach einer Optimierung zu messen ist der SpeedUp.

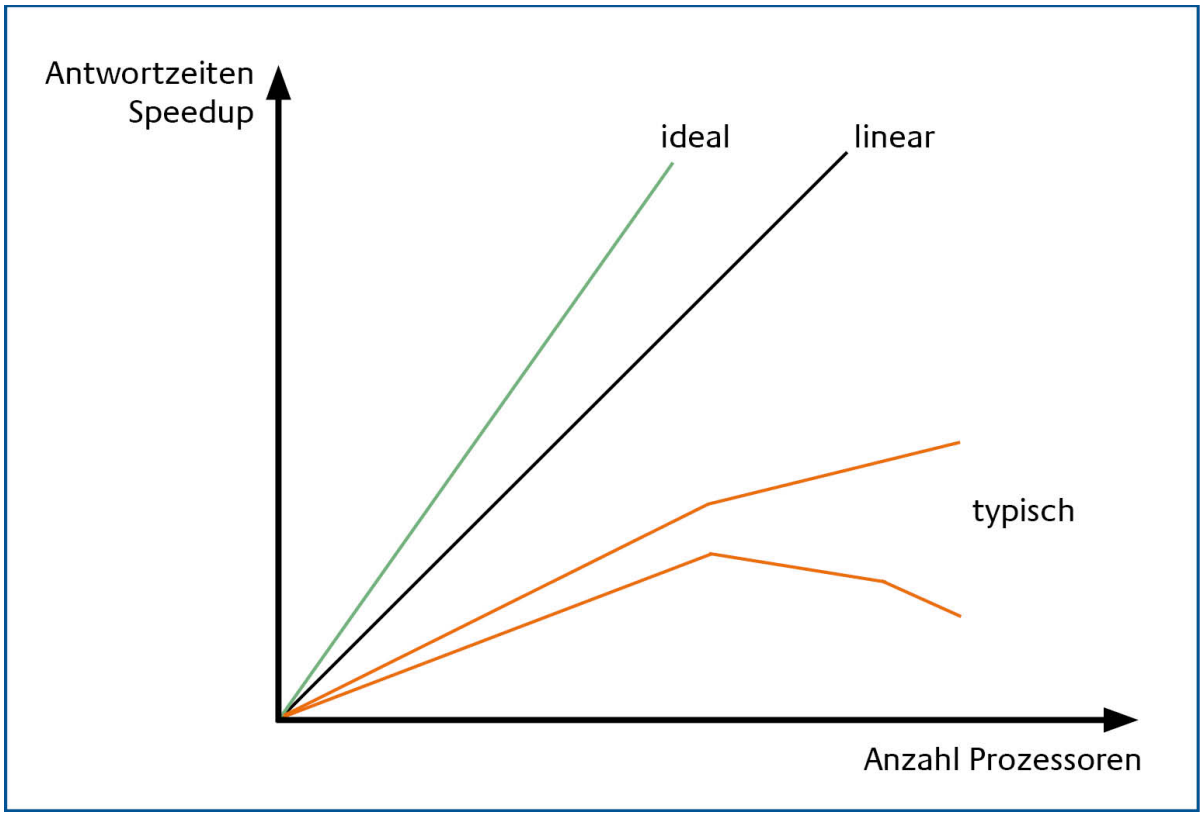


Abbildung 12: Speedup

Weiterhin unterliegen Parallele DBS verschiedenen Einflussgrößen. So entsteht durch das Starten oder das Beenden von Teiloperationen auf den verschiedenen Prozessoren ein „Overhead“. Ebenso können durch den Zugriff auf eine gemeinsam benutzte Ressource Wartezeiten (Interferenz) entstehen. Dies kann wiederum zu Sperrkonflikten (siehe Abschnitt „Verklemmungen (Log-Analyse)“) führen. Neben diesen Einflussgrößen existiert auch eine Varianz in der Ausführungszeit, die dadurch bedingt ist, dass die langsamste Teiloperation die Bearbeitungszeit einer parallelisierten Operation bestimmt (Skew).

Die bei den Parallelen DBS zugrunde liegenden Architekturen (Shared-Disk, Shared-Nothing, Shared-Erverything) wurden bereits im Abschnitt „Mehrrechner-Datenbanksysteme“ behandelt. Aufbauend auf diesen Architekturen ist es erforderlich, neben der Verarbeitungsparallelität auch E/A-Parallelität zu unterstützen. Eine Möglichkeit dieses zu erreichen, ist die Verwendung von Disk-Arrays. Dadurch kann das Leistungsverhalten gegenüber einzelnen Platten deutlich verbessert werden. Mit der E/A-Parallelität steigen allerdings auch die Anforderungen bezüglich der Fehlerbehandlung (z. B. Behandlung von Plattenfehlern). Um auch im Fehlerfalle alle durchgeführten Änderungen rekonstruieren und damit wiederherstellen zu können, ist es notwendig, dass alle Änderungen an Datenbankobjekten zu einer globalen Log-Datei zusammengeführt werden. Nur so stehen alle Änderungen, die über das gesamte Disk-Array erfolgt sind, in der richtigen chronologischen Reihenfolge zur Verfügung.

2.4 Betrachtung der Realisierungsalternativen

Bei der Entscheidung für eine bestimmte Mehrrechner-Architektur sprechen im HV-Umfeld viele Argumente für eine Realisierung in einer Shared-Disk-Architektur (SD). Denn während in Shared-

Disk-Systemen jede DBMS-Instanz Zugriff auf die gesamte physische DB besitzt, ist in Shared-Nothing (SN)-Systemen eine Partitionierung der Datenbank unter mehreren Rechnern erforderlich. Aus dieser grundlegenden Unterscheidung resultieren eine Reihe gewichtiger Vorteile (nach [Rahm94]) für SD:

- Die Systemverwaltung wird erheblich erleichtert, da die Bestimmung der Datenverteilung in SN-Systemen eine schwierige und aufwendige Aufgabe darstellt.
- Rechnerausfälle bleiben für SD ohne Auswirkungen auf die Externspeicherzuordnung, da die überlebenden Rechner weiterhin sämtliche Daten erreichen können. In SN-Systemen drohen dagegen Verfügbarkeitseinbußen, wenn die Daten eines ausgefallenen Rechners nicht mehr erreichbar sind. Ein einfacher Ansatz bei lokaler Verteilung sieht vor, dass ein überlebender Rechner die Partition des ausgefallenen Knotens vollständig übernimmt. Der Nachteil dabei ist jedoch, dass dieser dann mit hoher Wahrscheinlichkeit zum Engpass wird, da er die Zugriffe auf zwei Partitionen bearbeiten muss. Eine aufwendige Alternative besteht in der Replikation von Daten an mehreren Knoten.
- Eine Erhöhung der Rechneranzahl erfordert bei SN die Neubestimmung der Datenbank-verteilung, um die zusätzlichen Rechner nutzen zu können. Zudem ist das physische Umverteilen für große Datenbanken ein sehr aufwendiger Vorgang, der auch die Verfügbarkeit der betroffenen Daten beeinträchtigt. Diese Probleme entfallen bei SD, sodass sich Vorteile bezüglich der Erweiterbarkeit des Systems ergeben. Insbesondere sind keine Änderungen für den physischen DB-Aufbau beim Übergang vom Ein-Rechner- auf den Mehr-Rechner-Fall erforderlich, was wiederum eine starke Vereinfachung für die Administration bedeutet.
- In SN-Systemen bestimmt die DB-Verteilung für viele Operationen, wo diese auszuführen sind, unabhängig davon, ob diese Rechner bereits überlastet sind oder nicht. Damit bestehen kaum Möglichkeiten, die Auslastung der Rechner dynamisch zu beeinflussen. Vielmehr ist mit stark schwankender Rechnerauslastung zu rechnen, da zu einem bestimmten Zeitpunkt die einzelnen DB-Partitionen i. A. ungleichmäßig referenziert werden. Bei SD dagegen kann eine DB-Operation von jedem Rechner (DBMS) bearbeitet werden, da er Zugriff auf die gesamte Daten-bank besitzt. Dies ergibt ein weit höheres Potential zur dynamischen Lastverteilung und -balancierung. Damit können auch bei schwankendem Lastprofil ungleiche Rechnerauslastung sowie die damit verbundenen Leistungseinbußen umgangen werden.
- Die fehlende, statische DB-Aufteilung ergibt für SD auch erhöhte Freiheitsgrade zur Parallelisierung innerhalb *von Transaktionen*.

Für die SN-Architektur spricht wiederum, dass diese eine bessere Unterstützung für eine große Anzahl Prozessoren bietet. Weiterhin erfolgt in einer SN-Architektur die dedizierte Anbindung an einen Externspeicher mit einem Rechner. Daraus ergibt sich der Vorteil, dass für die Verbindung der Rechner untereinander, kein Hochgeschwindigkeitsnetzwerk erforderlich ist. Das Verbindungs-netzwerk wird lediglich für die Rechnerkommunikation benötigt und muss nicht zusätzlich die Kommunikation für die Externspeicherzugriffe transportieren, da diese direkt mit dem Rechner verbunden sind.

Basierend auf den Erkenntnissen der vorherigen Kapitel ist festzuhalten, dass hinsichtlich der HV-Anforderungen an Datenbanken diese - aus heutiger Sicht - am besten von lokal verteilten, integrierten Mehrrechner-DBS erfüllt werden. Dies gilt im Besonderen für die Anforderungen im Rahmen einer hochverfügbaren Architektur. Denn neben einer hohen Verfügbarkeit wird damit auch eine hohe Leistung und vor allen Dingen auch eine gute Skalierbarkeit erreicht. Ermöglicht

wird dies vor allem durch die Architekturklassen (siehe Abschnitt „Mehrrechner-Datenbanksysteme“) Shared-Nothing und Shared-Disk.

Die Tabelle 1 stellt die verschiedenen Aspekte zur Bewertung von Mehrrechner-DBS zusammenfassend dar.

	Parallele DBS (SD, SN)	***Verteilte DBS***	***Föderative DBS***	***Workstation/Server-DBS***
Hohe Transaktionsraten	++	0/+	0	0
Intratransaktionsparallelität	++	0/+	-/0	0/+
Erweiterbarkeit	+	0/+	0	0
Verfügbarkeit	+	+	-	0
Verteilungstransparenz	++	+	0	++
geografische Verteilung	-	+	+	0
Knotenautonomie	-	0	+	-
DBS-Heterogenität	-	-	+	-/0
Administration	0	-	-/--	0

Tabelle 1: Bewertung von Mehrrechner-DBS (nach [Rahm])

3 Replikation und Synchronisation

Die hochverfügbare Auslegung von Datenbanken verlangt neben der Redundanz der Hardware auch geeignete Verfahren zur Realisierung eines redundanten Datenbestandes.

Diese Redundanz kann durch Replikation der Datenbank oder durch deren Spiegelung erzielt werden.

Bei der Replikation handelt es sich um ein bidirektionales Verfahren, dass auch eine partielle Replikation definierter Datenmengen ermöglicht. Dabei unterscheiden sich die verschiedenen Ansätze zur Replikation hauptsächlich hinsichtlich der Datenmenge, die auf einmal repliziert wird, und den Zeitabständen, die zwischen den Speichervorgängen liegen. Diese Parameter haben einen wesentlichen Einfluss auf den möglichen Datenverlust bei einem Ausfall und auf die allgemeinen Leistungsparameter des DBMS.

Im Gegensatz dazu, stellt die Datenbankspiegelung ein unidirektionales Verfahren dar, bei dem keine partiellen Datenmengen gespiegelt werden, sondern ein kompletter Abzug der Datenbank übertragen wird.

Je nach Zielsetzung (Optimierung des Durchsatzes oder Minimierung der Risiken eines Datenverlustes) können die nachfolgenden Verfahren (oder Mischformen hiervon) eingesetzt werden.

3.1 Datenbankspiegelung

Bei der Datenbankspiegelung wird in festgelegten Intervallen, etwa jede Stunde oder jeden Tag, ein kompletter Abzug der Daten des DBMS auf ein (fernes) System überspielt. Dazu wird üblicherweise der normale Betrieb kurzzeitig unterbrochen, um die Datenintegrität zu wahren. Durch die Spiegelung wird im Wesentlichen ein normales Backup nachgebildet. Das ferne System wird in diesem Fall typischerweise als Cold-Standby betrieben und in einer Ausnahmesituation muss der Abzug zurückgespielt werden. In diesem Fall gehen maximal die geänderten Daten eines Sicherungsintervalls verloren. Daher sollte das Sicherungsintervall möglichst klein gewählt werden. Verkleinert man das Sicherungsintervall jedoch zu stark, so ist dieses Verfahren nicht mehr praktikabel, da die Datenverbindungen fast nur noch zur Übertragung der Daten verwendet werden. Daher ist dieses Verfahren nur in besonderen Ausnahmefällen oder in Kombination mit anderen Verfahren für HV-Ansätze geeignet.

Neben der Datenbankspiegelung sind noch weitere Varianten möglich. So bietet das Log-Shipping eine Möglichkeit die Log-Daten auf ein anderes System zu übertragen. Anhand dieser Daten werden alle Transaktionen des Primärsystems auf dem Sekundärsystem nachvollzogen. Dieses Variante hat allerdings den Nachteil, dass ein zeitlicher Verzug existiert, da die Log-Daten zuerst übertragen und anschließend ausgeführt werden müssen.

3.2 Synchrone Replikation

Bei der synchronen Replikation wird versucht, die Menge der übertragenen Daten zu minimieren und so zeitnah wie möglich die Daten auf dem Sekundärsystem sicher zu speichern. Zu diesem Zweck reicht das DBMS bei jeder atomaren Operation diese an das entfernte System weiter und führt sie gleichzeitig im eigenen System aus. Das primäre (aktive) System blockiert solange, d. h. es hält eine positive Bestätigung so lange zurück, bis es eine positive Bestätigung vom

Sekundärsystem erhalten hat. Dieses Verfahren bietet die größtmögliche Sicherheit vor dem Verlust von Daten, da bei einem Ausfall des aktiven Systems maximal die zuletzt bearbeitete Transaktion verloren gehen kann. Andererseits kann das Verfahren der synchronen Replikation erhebliche Auswirkungen auf das Latenzzeitverhalten der Datenbankanwendung haben, insbesondere wenn zur Übertragung der Daten große Entfernungen etwa in ein Backup-Rechenzentrum in einer anderen Stadt oder gar einem anderen Land zu überwinden sind.

Wird die synchrone Replikation eingesetzt, so ist darauf zu achten, dass die Synchronität auch für nachgeordnete Redundanzhierarchien erfüllt ist. Daher sollen verschiedene Synchronisationsansätze wie ROWA, Voting[2] oder das Primärkopien-Verfahren sicherstellen, dass eine durchgängige Synchronisation gegeben ist.

Es etwa nicht aus, dass die sekundäre Datenbank einen Abschluss der Schreiboperation signalisiert. Es ist darüber hinaus notwendig zu überwachen, ob die Informationen auch tatsächlich auf dem Permanentspeicher abgelegt wurden. Die Verwendung von flüchtigen Cache-Speichern ist nicht möglich, daher hat die Umsetzung des dargestellten Verfahrens Einbußen im Durchsatz zur Folge.

Bei der synchronen Replikation stellt außerdem die Wahrung der Datenintegrität eine Herausforderung dar, die von der Replikationssoftware überwacht und sichergestellt werden muss. Sollte beispielsweise ein Schreibvorgang auf einem System scheitern, so muss dieser solange wiederholt werden, bis er verlässlich abgeschlossen wurde, um keine Inkonsistenzen zu erzeugen.

3.3 Asynchrone Replikation

Bei der asynchronen Replikation erfolgen die Schreibvorgänge auf den redundanten Systemen asynchron. Das primäre System arbeitet unabhängig von der Bestätigung des Sekundärsystems und kann daher ohne eine Einschränkung der Performance oder der Latenzzeit arbeiten. Da die Schreibvorgänge auf dem Sekundärsystem in der Regel zeitverzögert erfolgen, erhöht sich dadurch die Anzahl der bei Ausfall der Primär-Datenbank verlorenen Transaktionen.

Neben den dargestellten Replikationsarten existieren ebenfalls Mischrealisierungen, die beispielsweise Schreiboperationen in (schnellen) Zwischenspeichern (Caches) ablegen, um diese periodisch an die Sekundärsysteme weiter zu leiten.

3.4 Zielkonflikte bei der Replikation

Bei der Auswahl der geeigneten Mechanismen ist grundsätzlich zwischen Durchsatz, Konsistenz und Ausfallsicherheit abzuwägen. Für die Replikation und Synchronisation ergeben sich demnach die drei Anforderungen: Verfügbarkeit, Konsistenz und Performance. Diese Anforderungen lassen sich jedoch nicht gleichzeitig erreichen. Änderungen zugunsten einer der Kriterien haben eine Minimierung der anderen zur Folge – deshalb spricht man von einem Zielkonflikt (siehe). Je öfter ein Datenelement vorhanden ist, desto größer ist die Verfügbarkeit. Dies erhöht die Fehlertoleranz (etwa bei Ausfall von Knoten). Bei Sicherung der Konsistenz verhält es sich entgegengesetzt, da hier bei Änderungen alle Replikate an das geänderte Datenelement angepasst werden müssen. Daher gilt hier: je weniger Kopien eines Datenelements bestehen, desto einfacher ist die Datenkonsistenz zu bewahren. Um die Performance zu steigern, kann man einerseits die Anzahl der Kopien erhöhen und so die Lesezugriffe beschleunigen, andererseits steigt mit der Anzahl der Kopien der Aufwand bei Änderungen, da alle Replikate geändert werden müssen. Vor der

2 Votieren

Beschreibung einer technischen Lösung bei der Replikation ist die Kenntnis der SOLL-Anforderungen notwendig, damit zugunsten einer höheren Verfügbarkeit ggf. eine Beeinträchtigung der Performance akzeptiert wird.

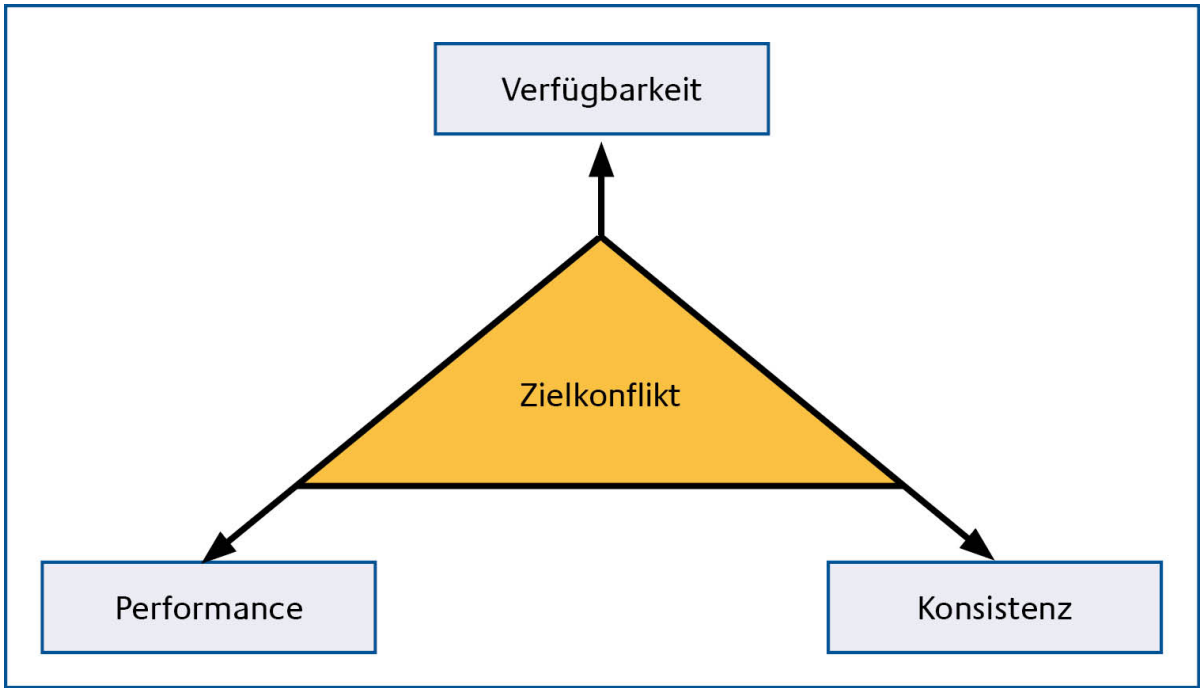


Abbildung 13: Zielkonflikte bei der Replikation und Synchronisation

3.5 Betrachtung der Realisierungsalternativen

Im Zusammenhang mit HV-Überlegungen ist eine synchrone Replikation theoretisch zwar erstrebenswert, jedoch können mit diesem Bestreben in der Praxis erhebliche Probleme verbunden sein. So sind mit der Umsetzung einer vollständigen Replikationstransparenz hohe Änderungskosten und eine geringe Skalierbarkeit verbunden. Bei komplexen, „global“ verteilten DB-Strukturen können weiterhin Verfügbarkeitsprobleme auftreten, da mit einer Steigerung der Komplexität auch eine Steigerung des zu replizierenden Datenvolumens einhergeht. Hinzu kommt, dass in einer „global“ verteilten DB-Struktur die Replikationsdaten über WAN-Strecken transportiert werden müssen. Aufgrund der Verzögerungen in der Kommunikation wegen der längeren Übertragungszeiten, können Replikationsfehler entstehen. Für einen Replikationszyklus steht nur ein bestimmtes Zeitfenster zur Verfügung, kann die Replikation in diesem Zeitfenster nicht abgeschlossen werden - weil z. B. eine Bestätigung des Replikationspartners innerhalb des Zeitfensters ausbleibt - wird sie von Neuem initiiert, obwohl der Replikationspartner diese bereits abgeschlossen hat. Diese Timingprobleme können dazu führen, dass der Replikationsmechanismus komplett neu initiiert werden muss.

Aufgrund dieser Eigenschaften haben sich in jüngster Vergangenheit Verfahren, die auf der Basis einer asynchronen Replikation (z. B. Schnappschuss-Replikation) arbeiten, als praktikabler erwiesen. Diese Verfahren lassen sich auch in einer HV-Umgebung zum Einsatz bringen, da sie auch eine Unterstützung z. B. für eine schnelle Katastrophen-Wiederherstellung bieten.

Neben den reinen Wiederherstellungs- und Replikationsverfahren, bietet sich auch die Möglichkeit der Verwendung von Push-/Pull-Verfahren an. Bei der Nutzung der Push-Replikation propagieren die Publisher[3] automatisch alle Updates an die Subscriber[4]. Bei der Nutzung einer Pull-Replikation prüfen die Subscriber selber, ob Änderungen vorhanden sind und holen diese dann ab. Die Pull-Replikation ist insbesondere bei einem Einsatz von mobilen Endgeräten interessant, um das Kommunikationsvolumen zu optimieren.

Eine wirkliche Alternative zu den genannten Replikationsverfahren kann nur in besonderen Ausnahmefällen eine Datenbankspiegelung sein. So ist es denkbar, dass eine Datenbankspiegelung in Ergänzung/Kombination mit den genannten Replikationsverfahren zum Einsatz kommen kann, z. B. um im Katastrophenfall eine schnelle Rücksicherung aus einem entfernten Rechenzentrum zu erlauben.

3 Publisher: Verbreiter von Änderungen
4 Subscriber: Empfänger/Nachfrager von Änderungen

4 Redundanz durch Datenbank-Cluster

Der Einsatz von Cluster-Architekturen wird in der Regel durch zwei Aspekte motiviert: der Verbesserung der Lastverteilung (Performance) und der Verbesserung der Verfügbarkeit. Beide Ziele können mit verschiedenen Cluster-Technologien erreicht werden, zu denen grundlegende Informationen dem Beitrag „Cluster-Architekturen“ im HV-Kompendium zu entnehmen sind. Die Verbesserung der Performance kann immer auch ein Teilaspekt beim Einsatz von Datenbank-Cluster sein; in diesem Kapitel liegt der Focus jedoch auf dem Einsatz von Datenbank-Clustern zur Wahrung hoher Verfügbarkeit. Dabei ist es wichtig zu bedenken, dass durch den Einsatz von Datenbank-Clustern die Verfügbarkeit des Datenbankdienstes erhöht wird, jedoch nicht die Verfügbarkeit der Daten. Hierfür sind andere Verfahren, insbesondere bei der Datenspeicherung (siehe Beitrag „Speichertechnologien“ im HV-Kompendium), erforderlich. Die übergreifende Eigenschaft aller Cluster-Architekturen ist die Entkopplung der Anwendung von der zugrunde liegenden Hardware. Hierbei lassen sich zwei grundsätzliche Verfahren unterscheiden, die in den folgenden Abschnitten dargestellt werden.

4.1 Hardware-Cluster

Weit verbreitet sind Cluster-Lösungen auf der Ebene der Hardware (1-in-M Cluster), da sie transparent für die zum Einsatz kommenden Anwendungen arbeiten. Dabei stellt ein Hardware-Cluster eine Gruppe von Serversystemen dar, die miteinander kooperieren und nach außen hin als ein einheitliches System auftreten. Die einzelnen Knoten bestehen aus - in der Konfiguration komplett voneinander unabhängigen - Rechnern, mit Spiegelung der Festplatten oder mit einer Verbindung über einen gemeinsamen, ausfallsicheren RAID-Array. Die Datenbankdienste laufen nur auf einem der Rechner. Die einzelnen Ausprägungen dieser Hardware-Cluster wurden bereits im Kapitel (Mehrrechner-Datenbanksysteme) beschrieben. Im Fall einer Störung werden die Dienste auf dem zweiten System gestartet, welche auf den gemeinsamen Datenpool zugreifen und so den Betrieb fortsetzen. Mit diesem Ansatz lassen sich Übernahmezeiten im Minutenbereich erreichen.

4.2 Datenbank-Cluster

Bei der Betrachtung von Hardware-Clustern fällt auf, dass es bei einem Ausfall eines Knotens, in jedem Fall zu einer Unterbrechung kommen muss, da nur ein Datenbankdienst aktiv ist und im Fehlerfalle ein anderer erst gestartet werden muss. Es ist kein übergeordneter Datenbankprozess vorhanden, der die Koordination eines gemeinsamen Datenzugriffs übernimmt. Damit können verschiedenste Probleme einhergehen, die auch beträchtliche Auswirkungen auf die Verfügbarkeit des Gesamtsystems haben können. So kann es beispielsweise beim Überschreiben des gleichen Datensatzes durch zwei Prozesse zu Inkonsistenzen in der Datenbank kommen. Daher kann im Fehlerfall die Zeit bis zur Übernahme eines ausgefallenen Knotens durch einen anderen Knoten zu lange dauern. Kürzere Übernahmenzeiten lassen sich nur dadurch realisieren, indem das DBMS eine geeignete Unterstützung für den Clusterbetrieb bietet. Durch das DBMS erfolgt eine Clusterbildung mit mehreren logischen Einheiten (Datenbank-Cluster).

Der Betrieb eines solchen Datenbank-Clusters kann in den beiden Varianten Aktiv/Aktiv (symmetrisch) oder Aktiv/Passiv (asymmetrisch) erfolgen. Es sind auch weitere Varianten möglich, z. B. ein Lastverteilungs-Cluster oder High-Performance-Cluster. Bei diesen Varianten wird die

Clustertechnologie nicht zur Verbesserung der Verfügbarkeit, sondern zur Verbesserung der Lastverteilung und zur Steigerung der Performance genutzt. Daher werden diese Varianten in diesem Kapitel nicht betrachtet.

Die beiden folgenden Kapitel betrachten die Auswirkungen von symmetrischen und asymmetrischen Cluster-Varianten bezüglich ihrer Bedeutung für die Hochverfügbarkeit der DBS.

4.2.1 Aktiv/Aktiv-Betrieb

Bei einem Datenbank-Cluster im Aktiv/Aktiv-Betrieb (siehe Abbildung 14) sind alle Knoten gleichzeitig aktiv. Fällt ein Knoten aus, so übernehmen die verbleibenden Knoten die Prozesse. Aufgrund dieser Architektur entstehen praktisch keine Ausfallzeiten; es kann aber zu Performanceeinbussen kommen. Daher ist bei der Planung dieser Zustand bereits zu berücksichtigen, um zu vermeiden, dass es zu einer Überlast bei dem Ausfall eines Knotens kommt und daraus weitere schwerwiegendere Probleme entstehen. Ein weiterer Vorteil dieses Ansatzes besteht in der Verteilung von Diensten auf mehrere aktive Knoten. Es können alle Knoten im Normalbetrieb genutzt werden und die Lasten werden über sie gleichmäßig verteilt. Damit kann ein sehr hohes Maß an Verfügbarkeit realisiert werden. Damit für den Anwender eine transparente Übernahme möglich ist, bedarf es jedoch einiger Funktionen, wie beispielsweise der automatisch erfolgenden Verbindungsaufnahme und Authentifizierung. Der Benutzer wird mit derselben Benutzer-ID angemeldet, die vor dem Ausfall verwendet wurde. So kann der Ausfall einer Session gegenüber dem Anwender vollständig transparent gehalten werden. Diese Funktionen sind bereits in Produkten namhafter Datenbankanbieter implementiert.

Die Vorraussetzung zur Nutzung eines Aktiv/Aktiv-Cluster im Rahmen einer DBS-Lösung ist allerdings ein entsprechend leistungsfähiges DBMS, welches diese Funktionen auch unterstützt.

Eine Lastverteilung, sowohl im Normalbetrieb als auch nach dem Ausfall von Knoten, kann in einigen DBMS mit entsprechenden Datenbankdiensten geschehen. Die Verteilungsverfahren können dabei nicht nur innerhalb bereits bestehender Instanzen genutzt werden, sondern auch bei neuen Instanzen, die bei Bedarf zugeschaltet werden.

Die Überwachung des Clusters kann mittels spezieller Cluster-Agenten erfolgen, die in der Regel Bestandteil des DBMS sind. Hierzu werden im Abschnitt „Monitoring und Audit“ weitere Informationen gegeben.

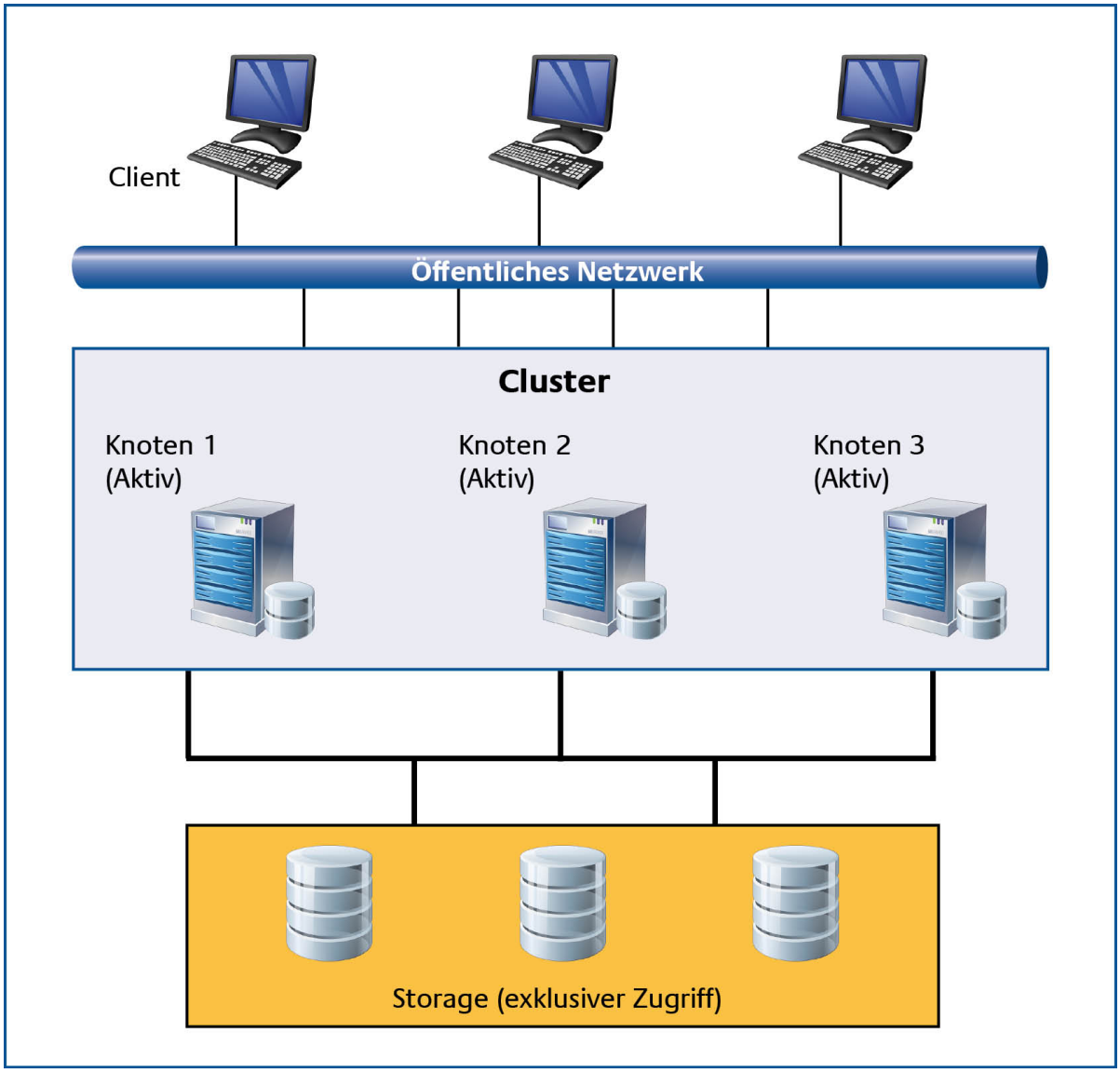


Abbildung 14: Aktiv/Aktiv-Cluster

4.2.2 Aktiv/Passiv-Betrieb

Neben dem Aktiv/Aktiv-Betrieb ist auch ein Aktiv/Passiv-Betrieb (siehe Abbildung 15) möglich. Bei dieser, auch als Failover-Cluster bezeichnete Variante, ist nur ein Knoten im Cluster aktiv. Bei einem Ausfall des aktiven Knotens wird ein anderer Knoten aktiv. Dabei wird der Status der Knoten überwacht und im Fehlerfall aktiviert eine geeignete Cluster-Software den zweiten Knoten. Hierzu werden zwischen den Cluster-Knoten Statusinformationen ausgetauscht, die für den Betrieb des Clusters notwendig sind. Die Kommunikation verläuft über ein dediziertes Netzwerk, das vom „Öffentlichen Netzwerk“ separiert ist. Die Trennung der beiden Netze ist erforderlich, damit Störeffekte (z. B. Timeout bei der Übertragung des Heartbeats[5] aufgrund starken Netzwerkverkehrs) beim Austausch der Statusinformationen vermieden werden.

Im Gegensatz zum Aktiv/Aktiv-Betrieb ist es in der Praxis jedoch oft nicht möglich, sofort den passiven Knoten in Betrieb zu nehmen. Denn die Übernahme vom aktiven auf den passiven Knoten bedingt in der Regel immer eine Rekonfiguration des Gesamtsystems. Das ist dadurch bedingt, dass

5 Periodisch ausgetauschtes Signal zwischen den Cluster-Knoten zur Überwachung des Status

aufgrund der Verzahnung verschiedener Prozesse und Anwendungen hier nicht ein einfaches Umschalten erfolgen kann, sondern es muss in der Regel die Anwendung auf dem übernehmenden Knoten hochgefahren werden. Dies ist auch der entscheidende Unterschied im Vergleich zu einem Aktiv/Aktiv-Cluster, denn bei diesem werden die Systeme bereits aktiv betrieben und sind daher sofort betriebsbereit. Für Anwendungen im Hochverfügbarkeits-Bereich geht bei dem Aktiv/Passiv-Betrieb durch die genannten (Re) Konfigurationsmaßnahmen wertvolle Zeit verloren, weshalb hier im Einzelfall genau zu prüfen ist, ob in einem Gesamtszenario noch eine ausreichend hohe Verfügbarkeit gewährleistet werden kann.

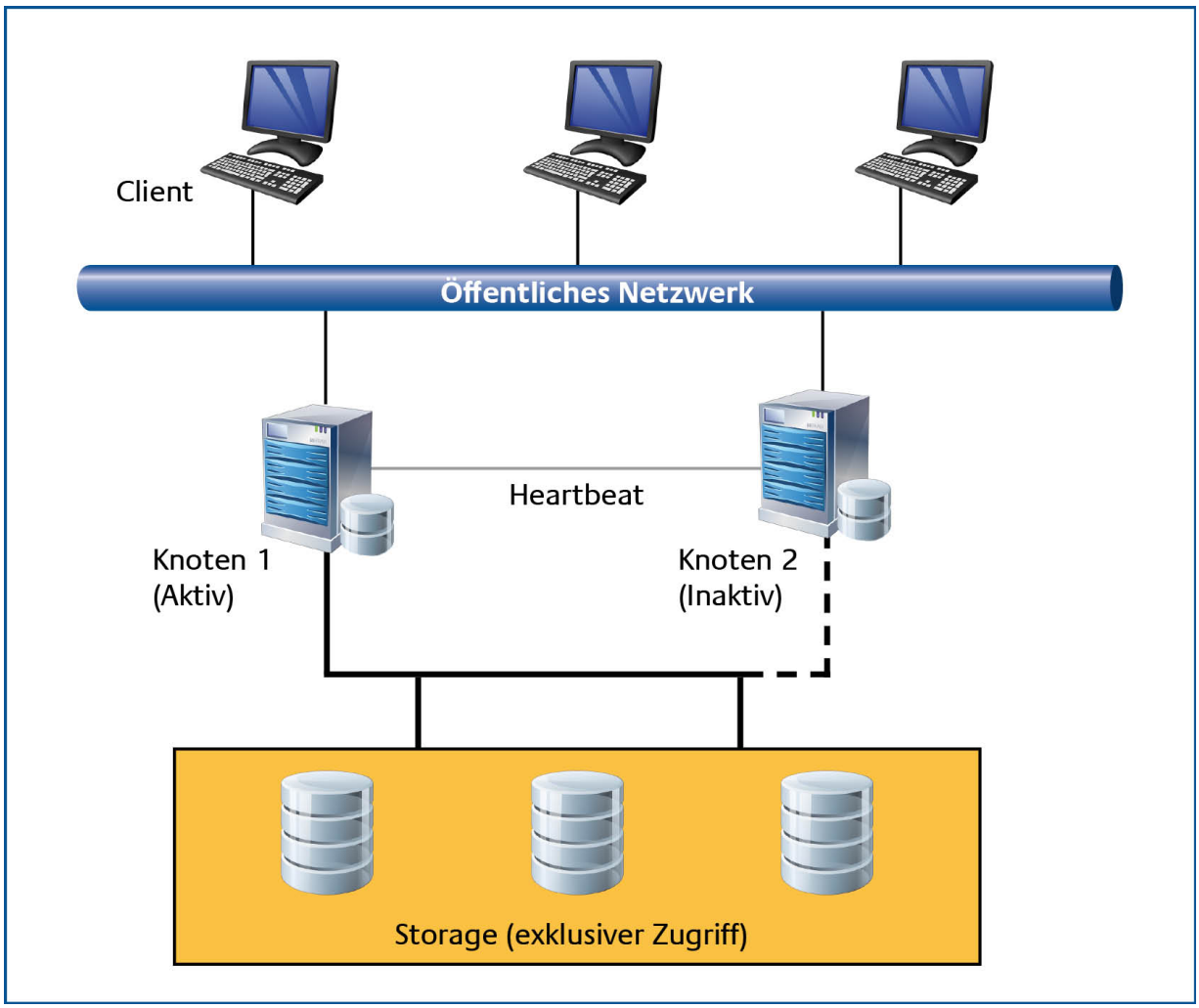


Abbildung 15: Aktiv/Passiv-Cluster (Failover-Cluster)

Geo-Cluster

Eine weitere Variante stellen Geo-Cluster (siehe Abbildung 16) dar, diese ermöglichen das Clustern von Datenbanken ohne geografische Grenzen. Geo-Cluster bieten, je nach Architektur, eine geografische Verteilung von 100 km und mehr. Es existieren bereits Lösungen, die Replikations-mechanismen über Kontinente hinweg unterstützten. Klassischerweise werden die Knoten auf verschiedene Rechenzentren geografisch verteilt. Fällt ein Rechenzentrum (z. B. durch Feuer oder Hochwasser) aus, können die verbleibenden Knoten weiterarbeiten.

So sind Geo-Cluster sehr gut geeignet, um eine hohe Verfügbarkeit auch im Fall des Eintretens schwer vorhersehbarer Ereignisse (insbesondere geografisch weiten Auswirkungen, z. B. Naturkatastrophen) zu gewährleisten. Die Funktionalität muss bei Bedarf, wenn sie nicht zum Lieferumfang des DBMS gehört, bei Drittanbietern eingekauft werden.

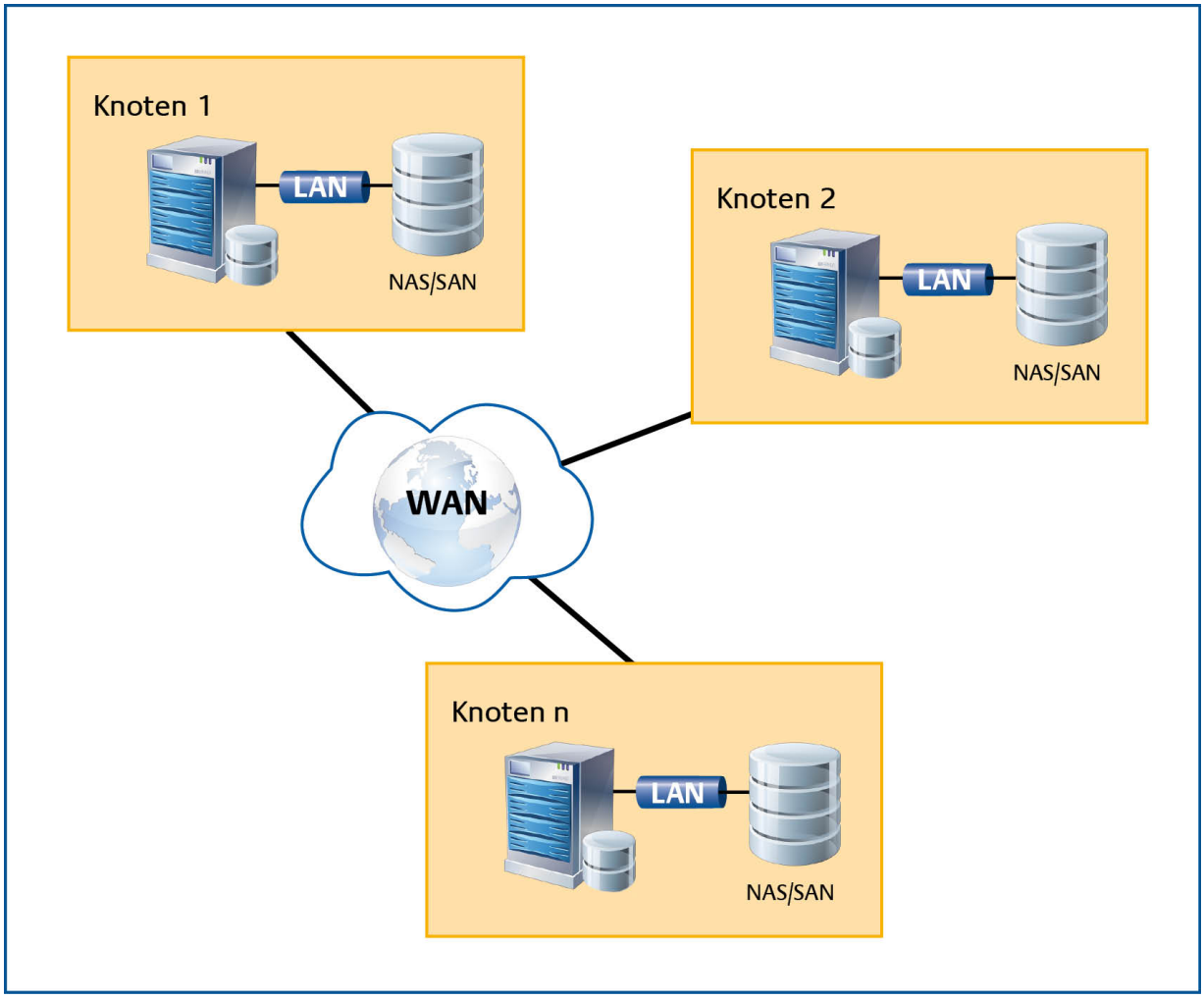


Abbildung 16: Geo-Cluster

4.3 Betrachtung der Realisierungsalternativen

Grundsätzlich verfolgen alle Clustervarianten das Ziel, durch die Erhöhung der Redundanz eine Verbesserung der Verfügbarkeit zu erreichen. Damit erfüllen sie grundsätzlich ein wesentliches Paradigma der Hochverfügbarkeit: die Beseitigung aller SPoF.

Damit stellen Cluster-Technologien geeignete Verfahren dar, um HV-Anforderungen gerecht zu werden. Insbesondere bei der Verwendung eines Aktiv/Aktiv-Clusters kann eine hohe Verfügbarkeit erreicht werden. Das Szenario, das ein Knoten ausfällt und damit die Gesamtlast auf die verbleibenden Knoten verteilt wird, ist allerdings mit in die Planungsphase einzubeziehen. Nur so kann vermieden werden, dass es zu Folgeeffekten durch Überlast des Gesamtsystems kommt.

Bei der Verwendung eines Failover-Clusters (Aktiv/Passiv) ist die Gesamtzeit für die Übernahmen des passiven Knotens bis zur Aufnahme des Wirkbetriebs zu berücksichtigen. Ein Aktiv/Passiv-Cluster ist nicht dafür ausgerichtet, mehrere Knoten gleichzeitig aktiv zu betreiben. Deshalb ist dieser nur eingeschränkt im HV-Umfeld einsetzbar.

Reine Hardware-Cluster verbessern zwar die Verfügbarkeit, allerdings fehlt ihnen eine übergeordnete Datenbanklogik, die in der Lage ist, Konflikte (z. B. Verklemmungen) oder andere datenbankspezifische Probleme selbstständig zu lösen. Hierdurch kann es zu Folgeeffekten kommen, die einen hochverfügbaren Einsatz vereiteln. Daher ist es auch an dieser Stelle notwendig, die konkreten Einsatzziele genauestens zu spezifizieren.

5 Standby-Datenbank

In hochverfügbaren DBS-Architekturen können verschiedene Verfahren zur Rechnerredundanz zum Einsatz kommen. Grundsätzlich lassen sich die beiden Verfahren: Hot-Standby und Cold-Standby, unterscheiden. Eine Standby-Datenbank ist eine transaktionskonsistente Kopie einer produktiven Primärdatenbank, auf einem zweiten, meist entfernten, Rechnersystem (Sekundärsystem). Es existieren verschiedene Verfahren um eine transaktionskonsistente Kopie herzustellen, wobei festzustellen ist, dass eine transaktionskonsistente Kopie nicht zwingend Hochverfügbarkeit bedeutet. Die beiden führenden Verfahren im Bereich der Standby-Datenbanken werden in den nächsten beiden Abschnitten dargestellt.

5.1 Hot-Standby

Bei der Nutzung eines DBMS im Standby-Betrieb wird ein Knoten aktiv betrieben (Primärsystem) und ein Knoten passiv (Sekundärsystem). Als aktiv wird ein Knoten dann bezeichnet, wenn über diesen Knoten Änderungen an den Daten von „außen“ möglich sind.

Das charakteristische Merkmal des Betriebs eines DBMS in einer Hot-Standby-Umgebung ist die Funktion des passiven Knotens; dieser kann im Parallelbetrieb mit laufen, nimmt aber keine Datenbankänderungen an (Ausnahme: Replikationsdaten). So verfolgt eine Realisierung im Hot-Standby-Betrieb das Ziel, eine zeitnahe Aktualisierung des passiven Sekundärsystems zu gewährleisten, um eine transaktionskonsistente Kopie des Primärsystems sicherzustellen. Nur dann ist eine verzögerungs- und verlustfreie Umschaltung vom Primär- auf das Sekundärsystem im Fehlerfall möglich. Dadurch, dass nur das Primärsystem aktiv ist, besteht die Notwendigkeit, die Änderungen des Primärsystems auf das Sekundärsystem zu übertragen. Hierzu können verschiedene Verfahren eingesetzt werden, die nachfolgend beschrieben sind.

5.1.1 Transaktionsaktualisierung durch Replikation

Eine Form der Transaktionsaktualisierung ist die Replikation eines laufenden Systems auf ein identisches, bereitstehendes Ersatzsystem (Sekundärsystem). Durch die Verwendung eines Replikationsmodells werden alle Änderungen des aktiven Knotens (Primärsystem) auf das Sekundärsystem repliziert. Im Rahmen einer Replikationstopologie können zwei unterschiedliche Verfahren zum Einsatz kommen, die nachfolgend in ihrer Anwendung bei einem DBS beschrieben sind.

Master/Master-Replikation

Bei der Verwendung einer Master/Master-Replikation sind beide Systeme, sowohl das Primär- als auch das Sekundärsystem als Master (siehe) konfiguriert.

Wenn ein Knoten für die Replikationsrolle Master eingerichtet ist, kann (muss aber nicht) er Änderungen von „außen“ (Anwendung, Benutzer) annehmen. Die Konfiguration bei einer Master/Master-Replikation definiert nur einen Knoten (Primärsystem) für die Kommunikation nach „außen“; dies ist der aktive Knoten. Das andere System (Sekundärsystem) kommuniziert nicht nach „außen“ und stellt das passive System dar. Demzufolge können Änderungen an der Datenbank nur auf dem Primärsystem durchgeführt werden. Hierzu ist eine entsprechende Konfiguration des DBMS notwendig, das die Kommunikation zu den Replikationspartnern steuert. Das Sekundärsystem arbeitet im Parallelbetrieb (passiv) und empfängt lediglich Daten, die im Rahmen der Replikation übertragen werden. Änderungen von „außen“, also durch den Anwender oder durch eine Anwendung, sind auf dem Sekundärsystem nicht möglich. Alle Änderungen, die auf dem

Primärsystem durchgeführt werden, werden zuerst in einem speziellen Datenbankbereich (z. B. Snappshot-Tabelle) gespeichert. Danach erfolgt eine Bestätigung (Commit) an das lokale DBS, so dass das Primärsystem sofort weiterarbeiten kann. Die Aufgabe des DBMS ist es nun, alle Änderungen der Snappshot-Tabelle auf das Sekundärsystem zu übertragen. Die Replikation erfolgt dabei entkoppelt von der Datenbankkommunikation des Primärsystems. Somit muss das Primärsystem nicht auf eine Bestätigung durch das Sekundärsystem warten, sondern kann sofort weiterarbeiten. Es ist die Aufgabe des DBMS, die Replikation der Änderungen zu steuern und zu überwachen. Fällt das Primärsystem aus, so wird durch das DBMS sichergestellt, dass das Sekundärsystem nun als aktiver Knoten verfügbar ist und auf diesem weitergearbeitet werden kann.

Dieses Verfahren gewährleistet eine hohe Verfügbarkeit und Transaktionssicherheit, da beide Systeme den gleichen konsistenten Datenbestand haben. Daraus resultiert der Vorteil, dass praktisch ohne Zeitverzögerung auf das Sekundärsystem umgeschaltet werden kann, wenn das Primärsystem ausfällt. Im HV-Umfeld ist dieses Verfahren ideal, da es neben einer hohen Ausfallsicherheit auch einen konsistenten Datenbestand garantiert. Allerdings bedarf es hierzu der Unterstützung entsprechender Mechanismen durch das DBMS, damit der Ausfall des Primärsystems erkannt und auf das Sekundärsystem umgeschaltet werden kann. Ein weiterer Vorteil der dedizierten Hot-Standby-Variante ist eine komplette Redundanz des DBS, also aller Ressourcen (z. B. Storage). Bei einer Clusterlösung, die auf ein gemeinsames Storage zugreift, müssen an dieser Stelle zusätzliche Maßnahmen etabliert sein, die einen SPoF ausschließen.

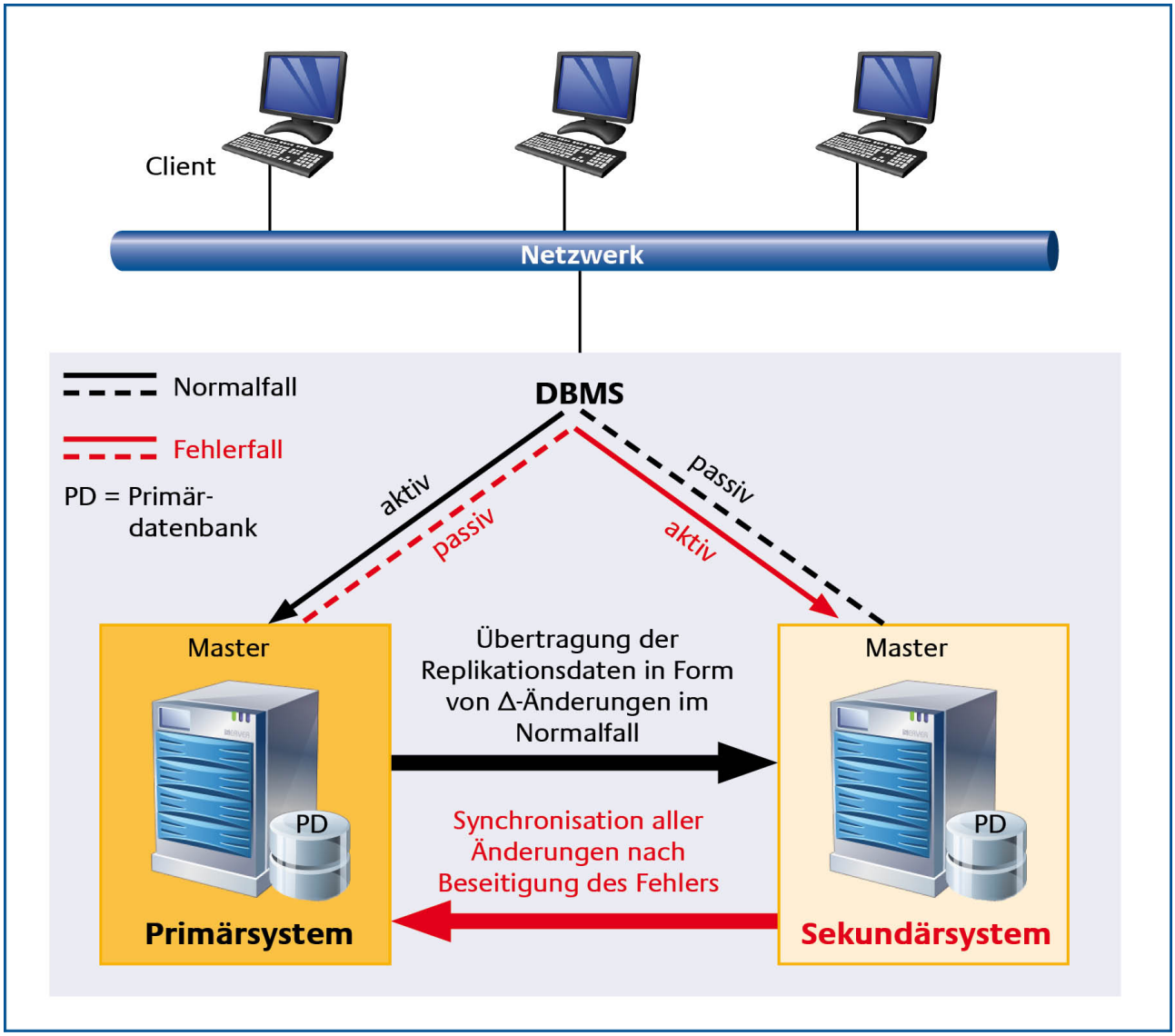


Abbildung 17: Replikation beim Master/Master Modell

Master/Slave-Replikation

Bei der Master/Slave Replikation wird das Primärsystem als Master und das Sekundärsystem als Slave konfiguriert. Hier erfolgt die Replikation der Änderungen wie bei der Master/Master-Replikation. Der Unterschied ist hier, dass bei Ausfall des Primärsystems das Sekundärsystem nur in der Replikationsrolle „Slave“ zur Verfügung steht. In einer Replikationsumgebung kann ein als „Slave“ konfigurierter Replikationspartner keine Änderungen von Datenbankanwendungen oder Datenbankanwendern entgegennehmen. Lediglich die Kommunikation mit den definierten Replikationspartnern ist möglich.

Durch das DBMS ist es nicht möglich diese Replikationsrolle zeitnah zu ändern. Daher genügt eine Master/Slave-Replikationstopologie - im Kontext des Einsatzes als Standby-System - nicht den Anforderungen, die an eine hochverfügbare Datenbank-Umgebung gestellt werden, und sollte nicht im HV-Umfeld verwendet werden.

5.1.2 LOG-Shipping

Neben den Replikationsverfahren kann auch über andere Verfahren sichergestellt werden, dass durchgeführte Änderungen vom Primär- auf das Sekundärsystem übertragen werden. Eine Möglichkeit ist durch die Übertragung von Log-Daten (Log-Shipping) gegeben. Da in Hochverfügbarkeitsszenarien oft eine hohe Zahl von Änderungen durchgeführt werden muss, erfolgt die Übertragung der Log-Daten in der Regel asynchron, um eine weitere Belastung des Primärsystems zu vermeiden. Die Log-Daten werden dazu genutzt, alle Änderungen auf einem Sekundärsystem nachzuvollziehen. Fällt die Primärdatenbank aus, können Anwender auf dem Sekundärsystem weiterarbeiten. Transaktionen, die auf dem Sekundärsystem ausgeführt werden, müssen dann später wieder auf die Primärdatenbank übertragen werden. Dieses Verfahren bietet ebenfalls gute Einsatzmöglichkeiten im Rahmen einer HV-Nutzung, allerdings ist dabei zu beachten, dass das Sekundärsystem immer mit einem gewissen Zeitversatz dem Primärsystem „hinterherläuft“. Im Fehlerfall können alle noch nicht übertragenen Transaktionen verloren gehen bzw. müssen erst nachgezogen werden (Logdateien stehen zwar zur Verfügung, aber das Roll-Forward[6], ist noch nicht durchgeführt).

5.2 Cold-Standby

Neben dem Hot-Standby-Betrieb, ist auch ein Betrieb nach dem Cold-Standby-Verfahren möglich. Im Cold-Standby-Betrieb ist ebenfalls nur ein Knoten (Primärsystem) aktiv. Im Gegensatz zum Hot-Standby-Betrieb wird das Sekundärsystem aber nicht zyklisch mit neuen Daten aktualisiert, sondern wird „offline“ in Bereitschaft gehalten. So kann im Fehlerfall nicht zeitnah auf das Sekundärsystem umgeschaltet werden, da dies nicht über den aktuellen Datenbestand verfügt. Weiterhin ist mit dem Einsatz auch ein hohes Risiko für den Verlust noch nicht abgeschlossener Transaktionen verbunden. Alle Informationen, die nicht auf einen externen Speicher (z. B. Archiv) übertragen wurden, sind bei einem Systemausfall verloren. Abhängig von der konkreten technischen Umsetzung kann es erforderlich sein, den aktuellen Datenbestand aus einem Archivsystem zurückzuspielen.

Bei einem Cold-Standby-Betrieb sind daher im Fehlerfall verschiedene Aktionen durchzuführen, z. B. Roll-Forward, um einen aktuellen Datenbankzustand auf dem Sekundärsystem zu erreichen.

Diese Eigenschaften des Cold-Standby-Betriebs lassen diese Variante für einen Einsatz in einer HV-Umgebung ungeeignet erscheinen. Eine Nutzung ist in einer HV-Umgebung nur in Kombination mit einer Cluster-Variante anzuraten.

5.2.1 Cold-Standby im Cluster

Das Cold-Standby-Verfahren kann auch in einer Cluster-Variante (siehe Abbildung 18) zum Einsatz kommen. Auch dann ist das Sekundärsystem solange nicht aktiv, bis das Primärsystem ausfällt. In diesem Verfahren sind beide Systeme typischerweise an einem privaten Netzwerk angeschlossen und überwachen die Heartbeat-Meldungen (siehe Abschnitt „Redundanz durch Datenbank-Cluster“). Solange das Primärsystem aktiv ist, werden über die Heartbeat-Verbindung Statusmeldungen an das Sekundärsystem gesandt; damit wird signalisiert, dass das Primärsystem aktiv ist. Bleiben diese Statusinformationen aus, so geht das Sekundärsystem davon aus, dass ein Problem vorliegt und versucht die Betriebsfunktionalität zu übernehmen. Der Zugriff auf Ressourcen (Storage, IP-Adressen, Dienste) ist dabei immer nur der aktiven Instanz gestattet. Im Fehlerfall startet sich das Sekundärsystem selber und versucht die Kontrolle über die Ressourcen zu erhalten. Da die beiden Knoten auf einen gemeinsamen Speicher zugreifen, verfügt auch der

6 Aktualisieren der DBS durch Einspielen aller zwischenzeitlich gemachten Änderungen.

passive Knoten über den gleichen Datenbestand. Hier ist lediglich eine Übernahme des primären Knotens erforderlich.

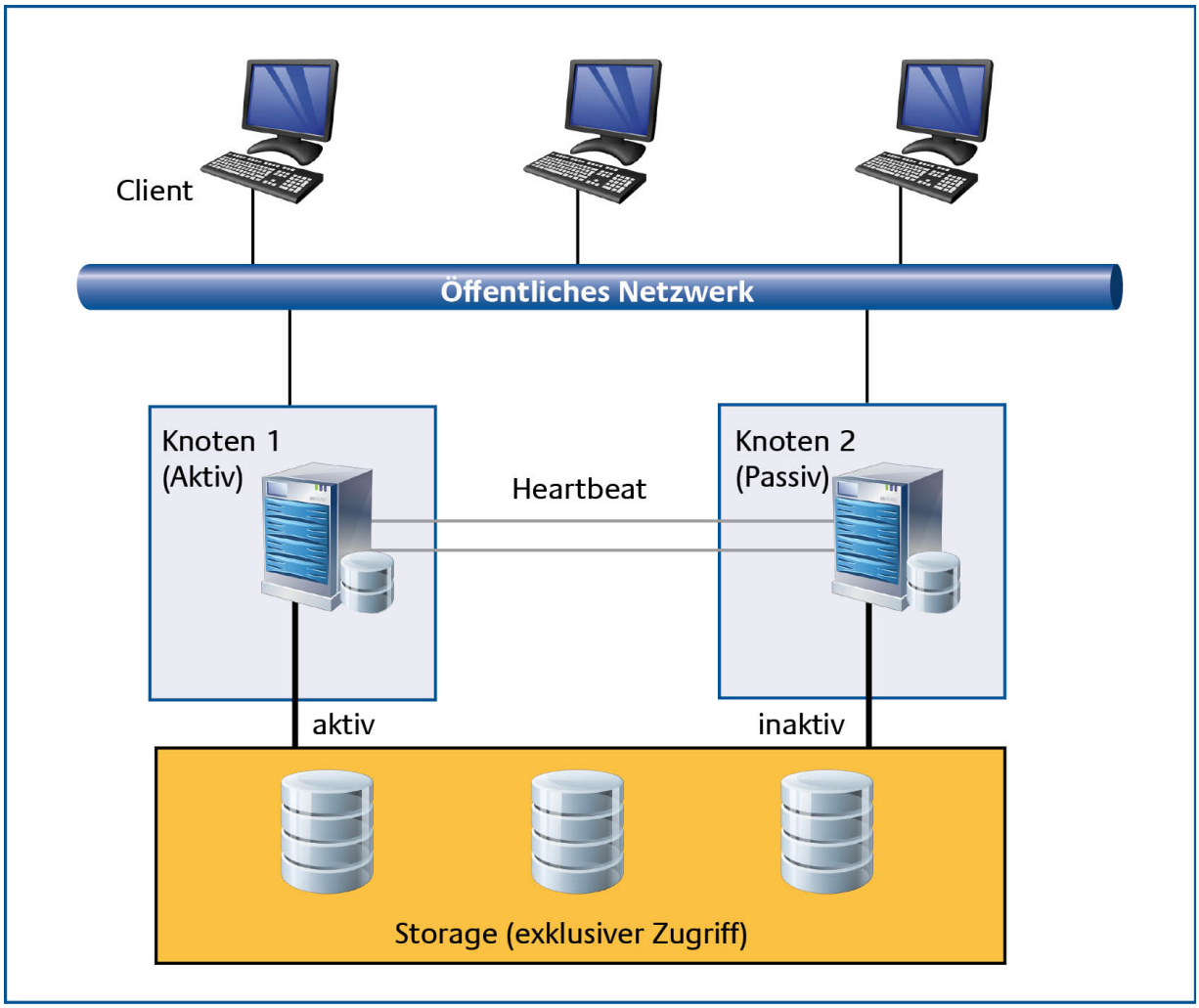


Abbildung 18: Schematische Darstellung Cold-Standby im Cluster

5.3 Betrachtung der Realisierungsalternativen

Die Intention des Standby-Betriebs im HV-Umfeld ist die Sicherstellung einer hohen Verfügbarkeit in Verbindung mit der Sicherstellung einer transaktionskonsistenten Datenbank, auch im Fehlerfall. Die besten Voraussetzungen für den Einsatz in einer HV-Umgebung lassen sich mit den Hot-Standby-Varianten erzielen. Ein dedizierter Cold-Standby-Betrieb erfüllt nicht die Anforderungen an hochverfügbare Systeme. Lediglich in Kombination mit einem Cluster kann hier eine ähnliche Verfügbarkeit erreicht werden.

Die Implementierung eines Hot-Standby-Systems in Kombination mit einer Master/Master-Replikation bietet gute Ansätze für eine hohe Verfügbarkeit. Auch eine Implementierung auf der Basis des Log-Shippings zeigt gute Hochverfügbarkeitseigenschaften. Allerdings ist hierbei zu beachten, dass das Sekundärsystem immer leicht verzögert auf dem Stand des Primärsystems ist. Hier gilt es zwischen Transaktionskonsistenz und der hohen Verfügbarkeit abzuwägen.

Grundsätzlich haben Standby-Implementierungen die aus HV-Sicht positive Eigenschaft, dass die DBS-Ressourcen redundant vorliegen. Das heißt, dass bei Ausfall eines kompletten Systems, ein zweites System diese Aufgaben komplett übernehmen kann. Aufgrund dieser Architektur sind auch bei einem Ausfall eines Systems keine Performance-Einbussen zu erwarten, da in beiden Fällen jeweils ein System arbeitet. Diese Eigenschaft ist aber bereits während der Planungsphase zu berücksichtigen, da eine Skalierung bei Standby-Systemen nicht gegeben ist. Wenn bereits während der Planungsphase damit zu rechnen ist, dass eine Lastverteilung auf verschiedenen Systeme notwendig werden kann, sollte man von vornherein eine Cluster-Lösung in Betracht ziehen.

Zur Wahrung der Hochverfügbarkeitsanforderungen ist in jedem Fall eine räumliche Trennung der beteiligten Systeme anzuraten. Minimal sollten die Systeme in getrennten Brandabschnitten untergebracht sein. Da die Systeme in der Regel rund um die Uhr laufen, sind sie den üblichen elektrischen Gefährdungen (Stromausfall, Blitzschlag, Kurzschluss) ausgesetzt und müssen auch dagegen gesichert werden.

6 Backup- und Recovery

Die Sicherung (Backup) und Wiederherstellung (Recovery) von Daten stellt eine wesentliche Säule im Rahmen der Hochverfügbarkeit dar. Nur über ein abgestimmtes Gesamtkonzept, das sowohl eine Sicherungsstrategie als auch eine Wiederherstellungsstrategie definiert, kann auch im Fehler-/ Katastrophenfall die Verfügbarkeit gewährleistet werden. Dabei hängen Sicherung- und Wiederherstellung unmittelbar voneinander ab und sind immer in ihrer Wechselwirkung zu betrachten. Da die Organisation einer Sicherungsstrategie äußerst komplex ist, empfiehlt sich eine sorgfältige Planung, beginnend mit einer umfassenden Anforderungsanalyse (Inventur). Auf der Basis der dadurch gewonnenen Erkenntnisse, werden mögliche Bedrohungsszenarien identifiziert und analysiert. Aufbauend auf diesen Informationen werden in einem nächsten Schritt die wichtigen (unternehmens- oder prozesskritischen) Daten identifiziert und deren Sicherung in einer Sicherungsstrategie festgelegt und dokumentiert. Als Orientierung können dabei die folgenden Schritte dienen:

1. Inventur
2. Bedrohungsszenarien analysieren
3. Wichtige Daten finden
4. Daten sichern
5. Dokumentation
6. Testen

Am einfachsten ist es, den Plan so aufzustellen, wie es im Idealfall (also ohne technische Einschränkungen) möglich wäre. Anschließend passt man den Idealplan an die reale Situation an, in dem man beispielsweise die Sicherungszyklen an die verfügbaren Zeitfenster anpasst. Steht der Plan und seine Implementierung, gilt es ihn zu testen, zu testen, zu testen und zu warten. Denn die zugrunde liegende Umgebung verändert sich und - ebenso wie Sicherheitsstrategien - muss der Plan stetig an die neue Situation angepasst werden. Daher empfiehlt es sich in der Regel alle sechs Monate, den Plan zu überprüfen und gegebenenfalls anzupassen.

Die Notwendigkeit von Sicherungs- und Wiederherstellungsstrategien sind gerade im HV-Umfeld gegeben. So sind z. B. Datenbank-Cluster in einer HV-Umgebung ein wichtiger Bestandteil der Gesamtarchitektur. Doch selbst bei einer Cluster-Lösung mit RAID-System können korrupte Datenblöcke auftreten (beispielsweise aufgrund eines Fehlers im RAID-Controller), die ohne Wiederherstellung aus einer Datensicherung nicht reparabel sind. Somit kann auch bei Nutzung der bereits beschriebenen HV-Technologien nicht auf Sicherungs- und Wiederherstellungs-Verfahren verzichtet werden.

Grundsätzlich kann die Sicherung einer Datenbank (Backup) entweder online oder offline erfolgen. Bei einer Offline-Sicherung ist der Zugriff auf die Datenbank, während der Sicherung gesperrt, wogegen er bei einer Online-Sicherung weiterhin möglich ist. Nachfolgend werden die beiden Varianten gegenübergestellt und bezüglich ihrer Eignung in einer HV-Umgebung untersucht. Daran schließt sich der Abschnitt „Wiederherstellung“ an, das sehr eng mit den beiden anderen Abschnitten verzahnt ist.

6.1 Online-Sicherung

Bei einer Online-Sicherung wird die Datensicherung während des Betriebes durchgeführt. Diese Art der Sicherung erfordert besondere Vorkehrungen, da während der Online-Sicherung die Instanz gestartet ist und somit Dateien geöffnet sind. Um diese Dateien im laufenden Betrieb kopieren zu können, muss das DBMS sicherstellen, dass für den Zeitraum der Datensicherung keine zu sichernde Daten verändert werden. Da aber bei hochverfügbaren DBS in der Regel immer Daten produziert oder geändert werden, werden diese Daten in einem anderen Bereich „zwischengespeichert“, unsichtbar für die Datensicherung, aber nutzbar für den Anwender. Ist die Datensicherung abgeschlossen, so schreibt die Datenbank die veränderten Sätze in die Dateien zurück.

Bei vielen Datenbanksystemen hat die Online-Sicherung einen weiteren Vorteil. Da die Veränderungs-Logs mitgesichert werden, ist eine "point in time"-Sicherung möglich. Dies bedeutet, die Datenbank kann zu einem beliebigen Zeitpunkt wiederhergestellt werden. Anwendungen ergeben sich z. B. nach fehlerhaften Eingaben oder Programmstörungen. Es gehen also nicht alle Änderungen bis zur letzten Sicherung verloren, sondern nur bis zur Störung.

6.2 Offline-Sicherung

Als Offline–Sicherung werden Sicherungen bezeichnet, die außerhalb des Benutzerbetriebes durchgeführt werden. Im HV-Umfeld ist oftmals ein 24/7-Betrieb erforderlich, weshalb diese Art der Sicherung nicht unmittelbar in HV-Umgebungen einsetzbar ist. Eine Möglichkeit, diese Art der Datensicherung dennoch „hochverfügbar“ durchzuführen, ist die Möglichkeit einer Offline-Sicherung in Verbindung mit einer Master-Slave oder auch Master-Master-Replikation. An diesem Beispiel wird deutlich, wie eng die einzelnen Detailaspekte voneinander abhängen und welche Auswirkungen diese auf das Gesamtszenario haben können. Grundsätzlich gilt die Aussage, dass für ein hochverfügbares DBS eine Online-Sicherung die bessere Variante ist.

6.3 Wiederherstellung

Mit der Erstellung einer Sicherungsstrategie sind unmittelbar auch die möglichen Varianten eines Wiederherstellungsszenarios verknüpft, sie sollten daher aufeinander aufbauen.

Unter dem Begriff Wiederherstellung (Recovery) werden sämtliche Maßnahmen verstanden, die notwendig sind, ein DBMS im Fehlerfall wieder konsistent herzustellen.

Eine der Herausforderungen bei der Durchführung einer Wiederherstellung ist die Sicherstellung der Transaktionskonsistenz. Gerade in komplexen, hochverfügbaren DBS Strukturen stellt dies eine nicht triviale Forderung dar. Zur Sicherstellung der Transaktionskonsistenz haben sich die ACID-Paradigmen etabliert.

Unter dem Akronym ACID werden die folgenden Forderungen verstanden:

- **A**: **Atomicity** (Atomizität)
  Eine Transaktion ist „atomar“, d. h.,. sie wird entweder **vollständig ausgeführt oder überhaupt nicht.**

- **C**: **Consistency** (Konsistenzerhaltung)
  Eine Transaktion überführt das System von einem **konsistenten** Zustand in einen anderen konsistenten Zustand.
- **I**: **Isolation**
  Die Transaktionen werden **isoliert** durchgeführt. Dadurch sind die Auswirkungen einer Transaktion erst nach ihrer erfolgreichen Beendigung für andere Transaktionen sichtbar.
- **D**: **Durability** (Dauerhaftigkeit)
  Die Dauerhaftigkeit der Transaktionen wird dadurch sichergestellt, dass erfolgreiche Transaktionen (commit) auf einem Plattenspeicher „verewigt“ werden, d.h. erfolgreich beendete Transaktion gehen nicht verloren.

Um den ACID-Anforderungen, auch im Rahmen einer Wiederherstellung zu genügen, bieten sich verschiedenen Verfahren an: Transaktions-Recovery, Crash-Recovery, Katastrophen-Recovery und Medien Recovery.

Eine Entscheidungshilfe, für welche typischen Fehlerarten sich welches Wiederherstellungsverfahren am besten eignet, kann der Fehler: Referenz nicht gefunden entnommen werden.

Wiederherstellungsverfahren	*Fehlerart*	*Auswirkungen*
Transaktions-Wiederherstellung	Transaktionsabbruch	auf eine Transaktion
Crash-/Katastrophen-Wiederherstellung	Systemausfall Stromausfall Überlastung	auf mehrere Transaktionen
Medien-/Katastrophen-Wiederherstellung	Block-Lesestörung Physikalische Zerstörung (z. B. Zerstörung eines RZ)	auf alle Transaktion

Tabelle 2: Fehlertypen und Wiederherstellungsverfahren

Die einzelnen Wiederherstellungsverfahren werden in den folgenden Kapiteln näher erläutert. Dabei veranschaulicht die die grundsätzlich beteiligten Systemkomponenten bei den Wiederherstellungsverfahren. Die in der aufgeführten Systemkomponenten haben dabei die folgende Bedeutung.

Permanente Datenbank

Der permanente Teil der Datenbank umfasst, neben der Datenbank, auch die Verwaltungsdaten (z. B. Kataloge, Adresstabellen, usw.).

Log-Puffer

Der Log-Puffer dient der Zwischenspeicherung der Änderungen.

Temporäre Log-Datei

Treten Transaktions- und/oder Systemfehler auf, so kann die Datenbank mit Hilfe der temp. Log-Datei wiederhergestellt werden. Dabei gilt:

DB + temp. Log = > DB

Archiv-Log/Archiv-Kopie

Kann z. B. aufgrund von Gerätefehlern nicht mehr auf die Datenbank zugegriffen werden, so kann die Datenbank aus der Archiv-Kopie und dem Archiv-Log wiederhergestellt werden. Dabei gilt:

Archiv-Kopie der DB + Archiv-Log => DB

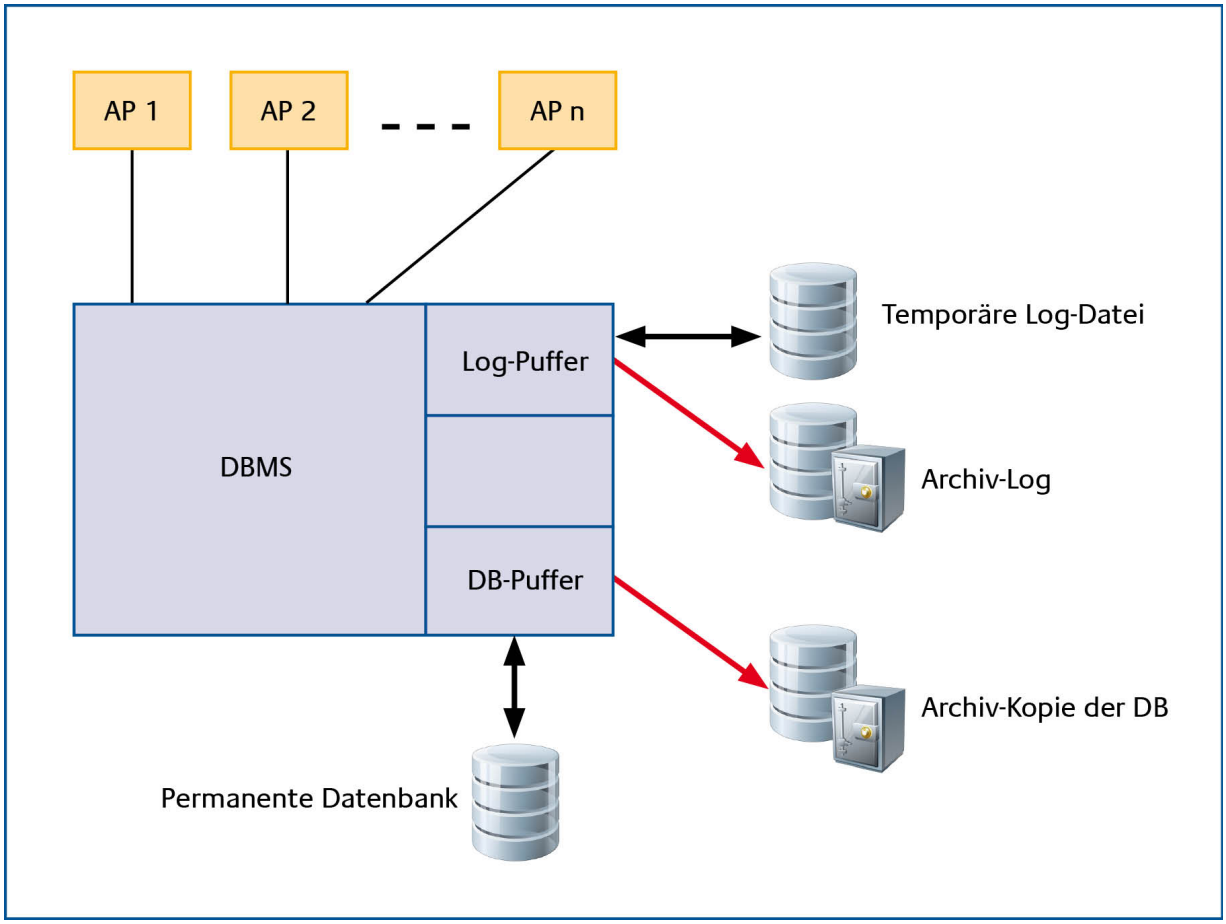


Abbildung 19: Beteiligte Systemkomponenten bei der Wiederherstellung (nach[Härder06])

6.3.1 Transaktions-Recovery

Durch ein Transaktions-Recovery soll die Atomarität der Transaktionen garantiert werden. Hierzu ist es allerdings notwendig, dass alle abgebrochenen Transaktionen auch entfernt (partiell oder vollständig) werden können. Daraus resultiert die Bedingung, dass wenn eine Transaktion abgebrochen wird, alle anderen Transaktionen auch abgebrochen werden müssen (Rollback), da dies sonst zu einem inkonsistenten Zustand führen kann.

Um die Atomarität sicherzustellen, ist ein Zurücksetzten einzelner, noch nicht abgeschlossener Transaktionen im laufenden Betrieb erforderlich. Hierzu kann entweder ein vollständiges Zurücksetzten auf den Transaktionsbeginn oder ein partielles Zurücksetzten auf einen Rücksetzpunkt (Checkpoint) innerhalb einer Transaktion erfolgen.

6.3.2 Crash-Recovery

Im HV-Umfeld kann durch ein Crash-Recovery versucht werden, die Atomarität und Persistenz von Transaktionen auch im Fehlerfall sicherzustellen. Dies kann zum Beispiel der Fall sein, wenn durch ein Betriebssystem- oder Anwendungsfehler eine/mehrere Transaktionen nicht vollständig ausgeführt werden konnten. Die damit möglicherweise einhergehenden Effekte derartiger

„korrupter“ Transaktionen dürfen nicht dauerhaft in die Datenbank gelangen bzw. dort verbleiben. Dies zu verhindern ist die Aufgabe eines Crash-Recoverys.

Hierzu lässt sich die Aufgabe des Crash-Recoverys in zwei Teilaufgaben gliedern. Zum einen wird ein partielles ReDo[7] aller erfolgreichen Transaktionen durchgeführt und damit kommt es zu einer Wiederholung verloren gegangener Änderungen. Zum anderen erfolgt ein UnDo[8] aller durch den Fehler unterbrochenen, „korrupten“ Transaktionen, diese werden damit aus der DB entfernt.

6.3.3 Medien-Recovery

Das Wiederherstellungsverfahren „Medien-Recovery“ dient hauptsächlich dem Schutz der Daten nach dem Eintreten von Schadensereignissen, wie beispielsweise Hardwarefehler, Brand oder Manipulation. Nicht immer lassen sich solche Ereignisse vermeiden oder deren Eintrittswahrscheinlichkeit als vernachlässigbar einordnen. Daher müssen entsprechende Maßnahmen umgesetzt werden, die im Eintrittsfall die Verfügbarkeit des DBS gewährleisten. Das Medien-Recovery, das auch als Langzeit-Recovery bezeichnet wird, baut auf den Backupverfahren auf, wie sie bereits in dem Abschnitt „Online und Offline Backup“ beschrieben sind.

Aufbauend auf den bereits genannten Sicherungsverfahren werden beim Medien-Recovery in regelmäßigen Abständen Checkpoints (Sicherungspunkte) „gezogen“. Derartige Sicherungspunkte haben den Vorteil, dass im Wiederherstellungsfall kein komplettes Backup zurückgespielt werden muss, sondern eine granulare Steuerung des Wiederherstellungszeitraumes möglich ist. Von diesem Sicherungspunkt aus kann die Arbeit der Transaktionen wieder aufgenommen werden. Dadurch gehen weniger Daten als bei einem vollständigen Rücksetzen verloren.

6.3.4 Katastrophen-Recovery

Bei einer Katastrophe handelt es sich um ein unvorhersehbares Ereignis (z. B. Erdbeben). Kommt es unvermittelt zu einem Katastrophenfall (also der vollständigen Zerstörung eines Rechenzentrums) so stellt das Katastrophen-Recovery in manchen Fällen die einzige Möglichkeit dar, die Verfügbarkeit der Daten wiederherzustellen.

Bei der Durchführung eines Katastrophen-Recoverys wird in der Regel auf eine Archivkopie zurückgegriffen, die an einem geografisch weit entfernten Ort vom Rechenzentrum aufbewahrt wird. Im Rahmen des Wiederanlaufs wird auf diese Archivkopie zurückgegriffen und die Verarbeitung an einem anderen Rechenzentrum fortgesetzt.

Dieses Verfahren unterliegt insbesondere der Einschränkung, dass sämtliche Transaktionen seit Erstellung der letzten Archivkopie verloren sind. Weiterhin ist in einer HV-Umgebung insbesondere das zeitaufwendige Einspielen der Archivkopie von Nachteil. Eine Alternative zur Wahrung der HV-Anforderungen stellt die Errichtung zweier (räumlich entfernter) Rechenzentren mit replizierter Datenbank dar. Dabei kann eine Replikation, z. B. gemäß Primary-Copy-Verfahren (Primär- und Sekundärkopien), erfolgen. Eine weitere Alternative für den Katastrophenfall stellt die Errichtung zweier gleichberechtigter (z. B. durch eine Master/Master-Replikation) Rechenzentren dar.

Eine weitere Variante, die im Rahmen des Katastrophen-Recovery zum Einsatz kommen kann, ist die Nutzung eines passiven Standby-Systems (siehe).

7 Informationen zum Nachvollziehen von Änderungen erfolgreicher Transaktionen
8 Informationen zum Zurücknehmen von Änderungen unvollständiger Transaktionen

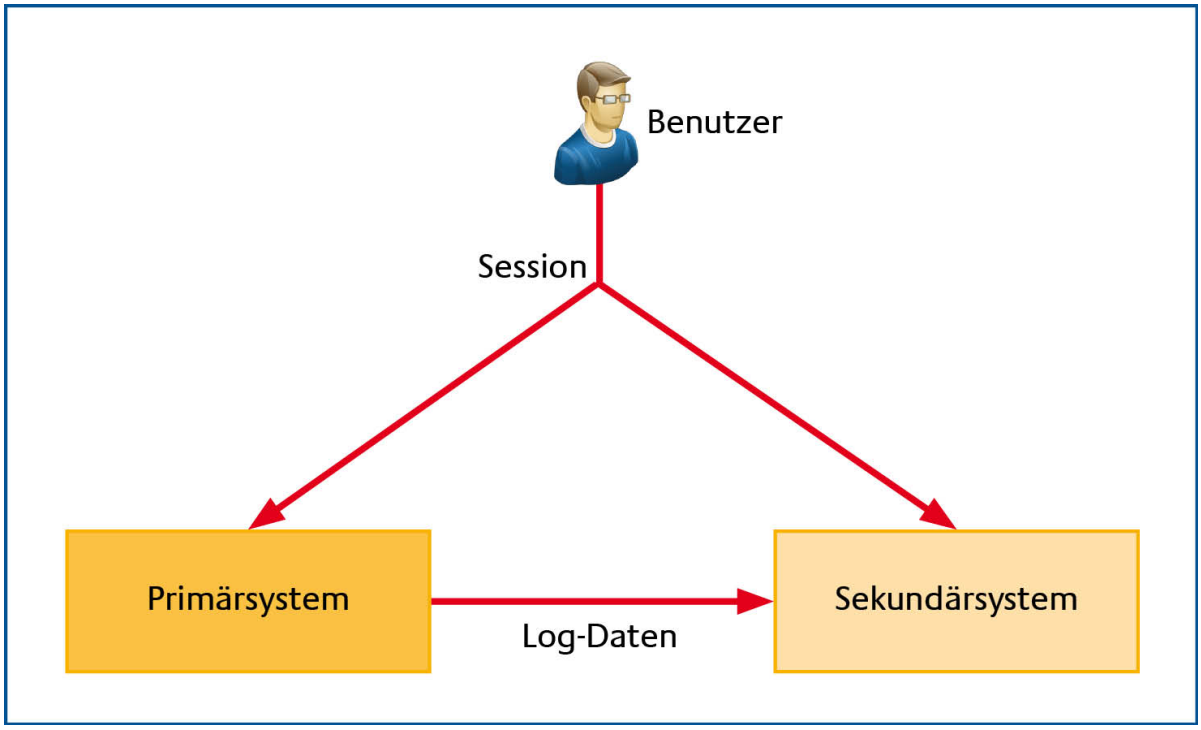


Abbildung 20: Passives Standby-System

Das passive Standby-System ist dadurch gekennzeichnet, dass alle Änderungen nur im Primärsystem erfolgen. Zur Erreichung von Lastverteilungsvorteilen kann bei (Lese-) Zugriffen das Sekundärsystem das Primärsystem entlasten, z. B. für Abfragen. Alle ReDo-Log-Daten werden sofort auf das Sekundärsystem übertragen. Im Fehlerfall (Katastrophenfall) gehen bei der Verwendung einer asynchronen Übertragung nur die Transaktionen verloren, die noch nicht auf das Sekundärsystem übertragen wurden.

Dies ist allerdings unter Umständen für wichtige Transaktionen nicht mehr tolerabel, daher empfiehlt es sich hier für wichtige (kritische) Transaktionen eine synchrone Übertragung der Log-Daten durchzuführen. Dabei ist zwischen den HV-Anforderungen und anderen Sicherheitsaspekten abzuwägen. Durch eine synchrone Übertragung kann es beim Sekundärsystem zu Verzögerungen kommen, da erst nach der Bestätigung (Commit) der Transaktion auf dem Sekundärsystem die Aktionen auf dem Primärsystem fortgesetzt werden können.

6.4 Betrachtung der Realisierungsalternativen

Für die im HV-Umfeld betriebenen DBMS stellen die für Wiederherstellungszwecke benötigten Synchronisations- und Logkomponenten sehr performancekritische Bestandteile dar. Sie können unter Umständen mehrere tausend Synchronisations- und Logaufrufe pro Sekunde betragen. So gilt es die Kriterien für die Auswahl des Wiederherstellungsverfahrens mit in die Planung für eine HV-konforme DBMS-Architektur einzubeziehen. In die Planung müssen das Antwortzeitverhalten des DBMS, sowie das zugrunde liegende Anforderungsprofil der zum Einsatz kommenden Anwendungen einbezogen werden. Nur durch ein - auf den Anwendungszweck und die konkrete Realisierung - abgestimmtes Sicherungs- und Wiederherstellungsverfahren kann den HV-Anforderungen auch in der Praxis gerecht werden.

7 Monitoring und Audit

Das Rückrad vieler Geschäftsprozesse stellen hochverfügbare Datenbanksysteme dar. Das Erreichen eines hochverfügbaren Zustandes stellt jedoch immer nur eine Momentaufnahme des aktuellen Status dar. Das Ziel muss es daher sein, diesen Status permanent garantieren zu können. Um dies zu erreichen, sind Maßnahmen zum Monitoring und Auditing unerlässlich. Eingebettet in eine proaktive Business-Continuity-Strategie gilt es wichtige Datenbankfunktionen und –prozesse kontinuierlich zu überwachen, um so eine effektive Störungsvermeidung zu erreichen. Durch die Überwachung festgelegter Datenbank- und Kommunikationsparameter können Konflikte rechtzeitig erkannt und behoben werden, bevor diese zu ernsthaften Fehlern führen können. Gerade bei komplexen Datenbanksystemen können kleinere Konflikte Folgeeffekte auslösen, die nur sehr aufwendig zu beseitigen sind. Während im Rahmen des Monitorings die Leistungsfähigkeit und Verfügbarkeit des DBS in Echtzeit geprüft wird, ergeben sich durch die Auditingfunktionen Aussagen zum Ressourcenzugriff (DBS).

Moderne DBMS müssen in der Lage sein, den aktuellen Systemzustand in Echtzeit zu ermitteln und darzustellen. Dabei sollten diese eine Ende-zu-Ende-Analyse durchführen, um auch mögliche Engpässe im Bereich der Infrastruktur (z. B. Netzwerk) zu erkennen.

Dazu enthält das DBMS einer aktiven Datenbank Mechanismen zum Erkennen von Ereignissen, zum Prüfen von Bedingungen und zum Durchführen bzw. Anstoßen von Aktionen. Bei allen Abfragen und Veränderungen, die in der Datenbank eintreffen, wird überprüft, ob eine Aktion durchgeführt werden soll. Diese Aktion kann wieder Veränderungen in der Datenbank auslösen oder eine Aktion außerhalb der Datenbasis anstoßen. Diese Mechanismen können für das Monitoring und Audit genutzt werden. So kann beispielsweise im Rahmen des Transaktions-managements der erfolgreiche Abschluss einer Transaktion (Commit) überwacht und protokolliert werden.

Im DBMS gibt es unterschiedliche Datenquellen, die für die Überwachung hilfreich sein können. Darunter fallen zum Beispiel die System-Logs, die das Hoch- und Herunterfahren der Datenbank, sowie Strukturänderungen aufzeichnen. Es gibt mehrere Möglichkeiten, Auditing in einem DBMS durchzuführen. Als Erste sei hier die Überwachung der Replikation genannt.

7.1 Überwachung der Replikation (Replikationsmonitor)

Welche Funktionen und Prozesse sinnvoll für eine Überwachung sind, hängt von der konkreten Implementierung des DBS ab. Neben der Überwachung allgemeiner Systemzustände ist in jedem Fall eine Überwachung besonders wichtiger Funktionen durchzuführen. Im Fall eines Datenbank-Clusters oder Standby-Systems ist dies zum Beispiel die Replikation. Der Replikation kommt bei diesen Implementierungen eine große Bedeutung zu, da sie die Transaktionskonsistenz der beteiligten Knoten sicherstellt. Ist die Replikation fehlerhaft oder kann nicht durchgeführt werden, ist die Verfügbarkeit des DBS gefährdet. So kann durch den Einsatz eines Replikationsmonitors die Replikation überwacht und deren Zustand ermittelt werden.

Jede Aktion seitens der Anwendung kann Fehlercodes zurückgeben, anhand derer z. B. Laufzeit-probleme erkannt werden können. Diese Fehlercodes geben der Anwendung die Möglichkeit, qualifiziert darauf zu reagieren und autonom Fehler zu bereinigen. In manchen Fällen erfordern aber Fehlerzustände eine zusätzlich Reaktionen auf der Serverseite. Das Erkennen solcher Fehlerzustände fällt in den Aufgabenbereich des Replikationsmonitors. Dieser überwacht

permanent den Zustand des Clusters, meldet Fehler und versucht Probleme sofort zu beseitigen, indem er, wenn möglich automatisch, Entscheidungen trifft. Der Replikationsmonitor ist ein Bestandteil des DBMS und wird individuell konfiguriert, da potenziell jede Anwendung spezifische Reaktionen auf Ereignisse benötigt. Einem Replikationsmonitor stehen mehrere Informationsquellen zur Verfügung, anhand derer die Funktionsfähigkeit des Clusters überwacht werden kann. Zunächst einmal muss der Monitor alle zu überwachenden Datenbankserver kennen. Diese Information bekommt er, indem er sich zunächst mit dem Master verbindet und sich von diesem alle Replikationsverbindungen geben lässt. Damit kennt der Replikationsmonitor jeden Server und dessen Rolle. Die einfachste Form der Überwachung besteht darin, dass der Monitor sich zyklisch den Status jeder Replikationsverbindung vom Master geben lässt und so den Replikationsstatus überwacht. Zusätzlich kann sich der Monitor auf allen Datenbankknoten anmelden, um direkt deren Verfügbarkeit festzustellen. Zweckmäßigerweise baut man IP-Verbindungen mit einem Keep-alive-Protokoll auf, um eine möglichst verlässliche Aussage zu bekommen. Speziell bei asynchroner Replikation kann es sinnvoll sein, die Differenz zwischen Master und Standby zu überwachen, um z. B. Überlastsituationen aufzuspüren. Dazu stehen in den Datenbanken Objekte zur Verfügung, anhand derer detaillierte Aussagen über den Zustand der Replikation gemacht werden können. Zur Verfügung stehen z. B. Transaktionszähler, Operationszähler (Operationen innerhalb einer Transaktion), höchste vergebene Objekt-ID und der Zeitpunkt der letzten Modifikation. Wenn keine offenen Transaktionen mehr vorhanden sind, müssen der Transaktionszähler und die höchste Objekt-ID zwischen Master und Standby gleich sein. Falls dies nicht der Fall ist, sind die Datenbanken nicht mehr synchron und es muss neu synchronisiert werden.

7.2 Verklemmungen (Log-Analyse)

Gerade in komplexen, hochverfügbaren DBS können durch die parallele Abarbeitung von Transaktionen Konflikte entstehen. Eine Art von Konflikten sind Verklemmungen (z. B. Deadlocks). Verklemmungen entstehen beispielsweise durch konkurrierende Transaktionen, die das gleiche Datenbankobjekt gleichzeitig ändern wollen, sich aber aufgrund unterschiedlicher Zugriffsberechtigungen gegenseitig blockieren. Ein derartiger Zustand verstößt gegen das Isolations-Paradigma von ACID und soll durch Sperrmechanismen abgefangen werden. Hierzu wird eine Sperrverwaltung genutzt, die durch Sperrtabellen sicherstellen soll, dass keine Verklemmungen entstehen.

Das führende Sperrprotokoll, das nahezu alle DB-Hersteller einsetzen, ist das 2-Phasen-Sperrprotokoll (2PL). Es kommt in der Praxis in einer Sonderform, dem strikten 2-Phasen-Sperrprotokoll, zum Einsatz. Bei der Verwendung des strikten 2-Phasen-Sperrprotokolls werden alle gesetzten Sperren erst am Ende der Transaktion (EOT) freigegeben. Dies verhindert das gegenseitige Beeinflussen von Transaktionen, führt aber häufig zu längeren Wartezeiten.

Die zur Anwendung kommenden Sperrprotokolle können grundsätzlich im Rahmen der Sperrverwaltung in zwei Varianten zum Einsatz kommen: der lokalen Sperrverwaltung oder einer globalen Sperrverwaltung. Beide Varianten werden nachfolgend beschrieben.

7.2.1 Lokale Sperrverwaltung

Für eine lokale Sperrverwaltung spricht vor allem die Performance, da die Verträglichkeit einer angeforderten Sperre mit bereits existierenden Sperren lokal entschieden werden kann. Dies minimiert die Kommunikation mit den dezentralen Sperrverwaltern erheblich. Für den Fall, dass ein lokaler Sperrverwalter ausfällt, sind lediglich die lokalen Sperren davon betroffen, alle anderen sind

davon unabhängig. Werden überwiegend lokale Transaktionen durchgeführt, bietet sich dieses Verfahren an. Werden überwiegend globale Transaktionen durchgeführt, so ist zu beachten, dass für jede globale Transaktionen zuerst eine Sperre vom zuständigen „lokalen“ Sperrverwalter angefordert werden muss. Daher ist in einem solchen Einsatzszenario zu prüfen, ob das Verfahren der lokalen Sperrverwaltung eine ausreichende Performance gewährleisten kann.

7.2.2 Globale Sperrverwaltung

Die globale Sperrverwaltung ist dadurch charakterisiert, dass alle Transaktionen alle Sperren von einer einzigen, festgelegten Station anfordern. Dies führt bei der Nutzung einer globalen Sperrverwaltung zu einem erheblichen Kommunikationsaufkommen.

Damit weist die globale Sperrverwaltung Merkmale eines SPoF auf. Wenn diese Station nicht mehr verfügbar ist, können Sperren nicht mehr aufgelöst werden, und dies kann zum Stillstand des DBMS führen. Nachteilig ist weiterhin die Tatsache, dass auch lokale Transaktionen ihre Sperren bei der zentralen Sperrverwaltung anfordern müssen. Damit geht auch der Verlust der lokalen Autonomie der Stationen verloren. Eine Globale Sperrverwaltung ist daher für eine HV-Umgebung nicht geeignet.

7.2.3 Verklemmungs-Erkennung

Basierend auf den Mechanismen der Sperrverwaltung kann eine Verklemmungs-Erkennung durchgeführt werden. Als Alternative kann auch die Strategie einer Verklemmungs-Vermeidung verfolgt werden. Grundsätzlich können Verklemmungen in komplexen DBS-Strukturen nicht ausgeschlossen werden und können immer auftreten. Daher ist die Wahl der richtigen Strategie sowohl für die Performance als auch für die Verfügbarkeit der DBS ein wichtiger Faktor.

Auf der Basis der Sperrinformationen (global/lokal) können drei unterschiedliche Mechanismen zum Einsatz kommen, die der Erkennung von Verklemmungen dienen. Die drei Verfahren sind nachfolgend dargestellt:

- Timeout-Verfahren
  Wenn eine Transaktion in einem bestimmten Zeit-Intervall keinen Fortschritt macht, wird sie zurückgesetzt.
- Zentralisierte Verklemmungserkennung
  Alle Stationen melden Wartebeziehungen an eine ausgezeichnete Station, die daraus einen globalen Wartegraphen (Wait-for Graph) erstellt.
- Dezentralisierte Verklemmungserkennung
  Jeder Transaktion wird eine "Heimat-Station" zugeordnet (gemeint ist die Station, wo die Transaktion gestartet wurde). Alle Stationen führen einen lokalen Wartegraphen.

7.2.4 Verklemmungs-Vermeidung

Der Ansatz der Verklemmungs-Vermeidung basiert im Wesentlichen auf Nutzung nicht-sperrbasierter Synchronisationsverfahren. Hier bieten zeitstempelbasierte Verfahren oder das Verfahren der optimistischen Synchronisation Ansätze zur Verklemmungsvermeidung. Der Ansatz der Optimistischen Synchronisation wird nachfolgend dargestellt.

Optimistische Synchronisation

Kennzeichnend für die optimistische Synchronisation ist die Aufteilung in drei Phasen:

- **Lesephase**
  Alle Operationen der Transaktion werden ausgeführt, aber alle gelesenen Daten werden in

lokalen Variablen der Transaktion gespeichert und alle Schreiboperationen werden zunächst auf diesen lokalen Variablen ausgeführt.

– **Validierungsphase**
  Hier erfolgt die Prüfung, ob die Transaktion möglicherweise in Konflikt mit anderen Transaktionen ist. Die Entscheidung erfolgt anhand von Zeitstempeln, die den Transaktionen in der Reihenfolge zugewiesen werden, in der sie in die Validierungsphase eintreten.

– **Schreibphase**
  Die Änderungen der Transaktionen, bei denen die Validierung positiv verlaufen ist, werden in dieser Phase in die Datenbank eingebracht.

7.3 Weitere Verfahren zur Überwachung

Nachfolgend sind weitere Verfahren, die im Rahmen einer Überwachung zum Einsatz kommen können, aufgeführt. In der Tabelle 3 werden die verschiedenen Varianten zusammenfassend dargestellt.

Aktive Mechanismen

Die aktiven Mechanismen der DBMS (z. B. Trigger) können für die Überwachung genutzt werden, indem jede Datenmanipulation einen Trigger auslöst, der geänderte Datensätze in eine Datei oder eine andere Datenstruktur schreibt.

Zeitstempelbasierte Überwachung

Jedem Datensatz kann ein Zeitstempel zugeordnet werden, der im Fall einer Änderung auf den Zeitpunkt der Änderung gesetzt wird. Anhand der Zeitstempel kann später entschieden werden, welcher Datensatz sich nach dem Zeitpunkt der letzten Extraktion geändert hat.

Ereignisbasierte Überwachung

Eine fortlaufende Überwachung der Datenobjekte und Services über eine feste Zeitdauer. Die ereignisbasierte Überwachung liefert detaillierte Informationen über Datenbanken, Tabellen, Deadlocks, Table Spaces, Buffer Pools, Connections oder Transaktionen.

Verfahren	*beobachtetes Sicherheitsmerkmal*	*Aufwand*
Replikationsbasierte Überwachung	Integrität	Gering
Log-Analyse	Integrität	Mittel
Aktive Mechanismen	Integrität	Hoch[9]
Zeitstempelbasierte Überwachung	Integrität	Gering
Ereignisbasierte Überwachung	Verfügbarkeit / Integrität	Hoch

Tabelle 3: Vergleich der Überwachungsmöglichkeiten

In der Software für die Funktionsübernahme können Überwachungssignale oder Keep-alive-Pakete zwischen Systemen verwendet werden, um die Verfügbarkeit zu bestätigen. Überwachungssignale werden durch eine ständige Kommunikation von Systemservices zwischen allen Knoten eines

9 Hoher Zeitaufwand bei Entwicklung und zur Laufzeit

Clusters realisiert. Wenn kein Überwachungssignal erkannt wird, beginnt die Funktionsübernahme durch ein Ausweichsystem. Endbenutzer bemerken gewöhnlich nicht, dass ein System ausgefallen ist.

7.4 Betrachtung der Realisierungsalternativen

Für DBS im HV-Umfeld ist sicherzustellen, dass diese auch beim Auftreten von Fehlern oder Anomalien funktionsfähig bleiben. Hierzu wurden in den letzten Jahren Verfahren und Prozessregeln[10] entwickelt, die sicherstellen sollen, dass durch Anwenden geeigneter Metriken in Kombination mit einem pro-aktiven Continuity-Management rechtzeitig mögliche Probleme erkannt und vor dem Eintreten eines wirklichen Fehlers beseitigt werden können. Die Informationsbasis hierzu liefern die überwachten Systemfunktionen. Der Festlegung der Überwachungsparameter kommt dabei eine große Bedeutung zu. Die Möglichkeiten sind hier sehr vielschichtig und hängen von der konkreten Implementierung ab. Für Clustersysteme sollten mittels spezieller Cluster-Agenten wichtige Komponenten des Clusters und auch der Anwendungen überwacht werden. Diese Agenten sind meist Bestandteil des DBMS. Es können auch Agenten von Cluster- oder Monitoring-Software verwendet werden. Die Agenten sollten so konfiguriert sein, dass sie sowohl die Verfügbarkeit als auch die Integrität von DB-Prozessen und –diensten überprüfen. Wird ein Fehler erkannt, so können die Agenten beispielsweise automatisch einen Failover und Neustart initiieren.

Während die praktizierte Überwachungsstrategie nur einen mittelbaren Einfluss auf die Verfügbarkeit hat, wirken sich Entscheidungen im Rahmen der Vermeidung von Verklemmungen unmittelbar auf die Verfügbarkeit des DBS aus. So kann durch die Nutzung einer Globalen Sperrverwaltung ein SPoF entstehen, der unter Umständen ein komplettes DBS zum „Erliegen“ bringt. Eine derartige Entscheidung muss daher genau auf den Einsatzzweck abgestimmt werden. Neben diesen Überlegungen hat sich in der Praxis als zugrunde liegendes Sperrverfahren das strikte[11] 2PL Verfahren durchgesetzt.

10 ITIL, Cobit

11 das strikte 2 PL ist eine Sonderform des 2PL

8 Virtualisierung

Die in den vorherigen Kapiteln betrachteten Verfahren basieren ausnahmslos auf eine eindeutige Zuordnung von Anwendungen zu physikalischen Systemen (Ressourcen). Im Rahmen der Virtualisierung findet hier eine Entkopplung statt, d. h. eine Anwendung (DBMS) allokiert nicht ein ganzes Serversystem oder die zugeordneten Ressourcen, sondern erhält die notwendigen Ressourcen „virtuell“. Neben Vorteilen, einer Virtualisierung im Bereich QoS, bieten Virtualisierungstechniken auch Ansatzpunkte zur Steigerung der Verfügbarkeit von DBS. Daher werden zunehmend „Virtualisierungstechniken“ im HV-Bereich interessant. Zu beachten ist jedoch, dass mit der Einführung von Virtualisierungstechniken (z. B. Servervirtualisierung) auch die Komplexität zunimmt. So ist zu prüfen, ob die zum Einsatz kommende Virtualisierungstechnik für den Einsatz in einem HV-Umfeld geeignet ist. Werden Virtualisierungstechniken zur Konsolidierung mehrerer physischer Systeme auf logische Systeme genutzt, so ist in diesem Fall sicherzustellen, dass dadurch kein SPoF entsteht der die Verfügbarkeit gefährdet.

Daher ist die Nutzung von Virtualisierung abhängig vom Grad der geforderten Hochverfügbarkeit und der zum Einsatz kommenden Gesamtarchitektur. Im Umfeld einer Datenbank gibt es mehrere Verfahren, welche dem Prinzip der Virtualisierung folgen. So können beispielsweise Verzeichnisse, IP-Adressen oder DB-Server virtualisiert werden. Diese Verfahren haben als gemeinsames Ziel, die Verfügbarkeit der Dienste und Daten zu gewährleisten.

8.1 Virtuelle Verzeichnisse

Ein Ansatz, die Verfügbarkeit eines DBS durch Virtualisierung zu verbessern, liegt in der Nutzung Virtueller Verzeichnisse. In der Regel erfolgt die Konfiguration und Anbindung eines DBS durch die Verwendung fester Verzeichnisangaben oder Mountpoints. Wird für die Verwendung der DBS eine feste Verzeichnisstruktur gemounted, ist diese nur nutzbar, solange die Ressource verfügbar ist. Durch die Bereitstellung von Virtuellen Verzeichnissen erfolgt der Zugriff auf Ressourcen nicht über eine eindeutige Verzeichniszuordnung, sondern über einen Aliasnamen[12]. Der Einsatz von virtuellen Verzeichnissen entkoppelt die direkte Zuordnung zwischen DBS und Speicherort. Das DBS greift mittels eines Aliasnamen und nicht mehr mit einer konkreten Adresse auf die Ressourcen zu. Dabei verhält sich die Darstellung des Aliasnamens des Verzeichnisses für das DBS vollkommen transparent. Diese Art der Virtualisierung verschleiert die physikalische Verzeichnis-/ Ressourcenstruktur zugunsten einer flexiblen Anbindung. Dadurch ist es nicht mehr erforderlich, z. B. bei Veränderungen an der Verzeichnisstruktur, Änderungen am DBS durchführen zu müssen, solange sich der Aliasnamen nicht ändert. Virtuelle Verzeichnisse können zur Verbesserung der Verfügbarkeit beitragen, da ein Verzeichnisbaum nicht mehr aus einer, sondern aus verschiedenen Ressourcen, gebildet werden kann. Fällt eine Ressource aus, kann auf den verbleibenden weiter gearbeitet werden.

In der Praxis haben sich zwei Arten Virtueller Verzeichnisse etabliert: das Distributed File System (DFS) und das Network File System (NFS). Beiden gemeinsam ist die Bereitstellung eines „virtuellen“ Verzeichnisses für Anwendungen oder Benutzer, das durch mehrere physikalische Server bereitgestellt wird (siehe auch „Speichertechnologien im HV-Umfeld“). Virtuelle Verzeichnisse tragen auf diese Weise dazu bei SPoF zu vermeiden.

12 Alias: Virtueller Verzeichnisname

8.2 Virtuelle IP-Adresse

Die Virtuelle IP-Adresse (VIP) bezeichnet die dynamische IP-Adresse, die im Zusammenhang mit einer Cluster-Lösung auftritt. Alle Rechner im Cluster-Verbund haben individuelle Adressen, während die virtuelle IP-Adresse zusätzlich an aktive Knoten „ausgeliehen“ wird. Von außen wird der Cluster immer über die virtuelle IP-Adresse angesprochen, sodass automatisch die aktiven Systeme erreicht werden (siehe auch den Beitrag „Netzwerk“ im HV-Kompendium, Abschnitt „Aktive Netzkomponenten“).

Im Fall eines Failover, bei einem Aktiv-/Passiv-Cluster, wird ein Standby-System aktiviert und erhält nun die virtuelle IP-Adresse. Damit werden alle weiteren Verbindungen mit dem neu aktivierten System hergestellt, sodass es die Funktion des ausgefallenen Knotens übernimmt.

Durch die Einrichtung virtueller IP-Adressen ist eine Virtualisierung von DB-Diensten möglich. Die Dienste werden mit Hilfe der Clusterarchitektur ausfallsicher und skalierbar bereitgestellt. Dabei wird die direkte Abhängigkeit zwischen Applikation und spezifischem DB-Server aufgehoben. Aus Sicht der Applikation kann über die Virtuelle IP-Adresse der benötigte Dienst auf einem beliebigen aktiven Cluster-Knoten abgerufen werden. Virtuelle IP-Adressen sind Standard bei der Nutzung von Cluster-Lösungen und bei der Anbindung an Speichernetzwerken wie beispielsweise SAN.

8.3 Virtuelle Datenbanken

Bei der Betrachtung „virtueller Datenbanken“ muss grundsätzlich zwischen logischer und physischer Ebene unterschieden werden (siehe ANSI-SPARC 3-Schema-Konzept im Abschnitt „DB-Architekturen“). Eine virtuelle Datenbank auf der logischen Schicht besteht aus individuellen Tabellensätzen, Sichten und gespeicherten Prozeduren mehrerer Datenquellen. Für die aufrufende Anwendung erscheint eine virtuelle Datenbank wie eine einzelne Datenbank. Die aufrufende Anwendung kann jedoch nicht auf die zugrunde liegenden Datenquellen zugreifen, wodurch der Zugang auf andere Unternehmensdaten verhindert wird (siehe). Der Anwender erhält nur den Zugriff auf die Daten, die für die Erledigung seiner Aufgaben innerhalb des Gesamtprozesses erforderlich sind. Nach dem Prinzip der Separation werden die Daten aus dem gesamten Datenpool spezifisch für Anwender und Anwendung zugewiesen. Auf der physischen Ebene wird die Virtualisierung hardwarenah realisiert. Durch Virtualisierung des Speichers (siehe Beitrag „Speichertechnologien“ im HV-Kompendium), des Netzwerks und der Server kann mit Hilfe einer Virtualisierungsschicht der Ausfall einzelner Komponenten für den Anwender transparent kompensiert werden.

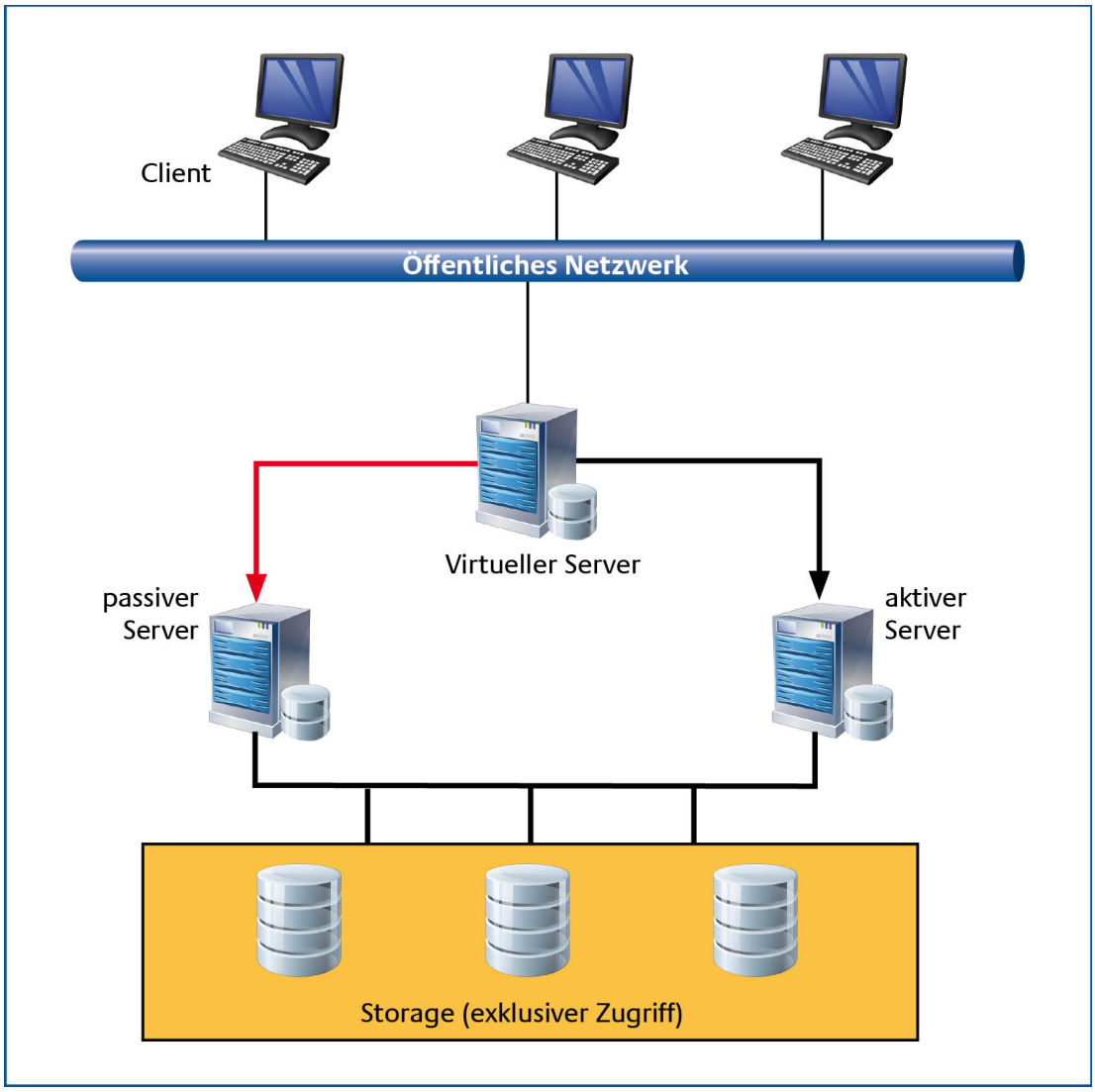


Abbildung 21: Virtueller Server im Cluster

Aus vielen physikalischen (auch verstreut installierten) Servern wird im Bedarfsfall ein logischer Server. Die systemübergreifende Administration ermöglicht es, Prozessoren oder deren Logikbereiche, Festplatten, Systembusse, Memory und I/O-Kanäle in Pools zu bündeln und, neu kombiniert und optimiert, für den Bedarf der Datenbank bereitzustellen. Allerdings verändert die DB-Virtualisierung auch die Komplexität des Gesamtsystems. Durch die Einführung der Virtualisierung ergibt sich eine zusätzliche Betrachtungsebene, die eine Neubewertung anderer Ebenen erforderlich macht. So ergeben sich veränderte Schnittstellen und Abhängigkeiten insbesondere zu den Bereichen Storage, Server und Netzwerk. Dies ist in jedem Fall mit in die Gesamtbetrachtung einzubeziehen. Die Hersteller namhafter Datenbank- und Virtualisierungssoftware bieten Produkte an, welche die unterschiedlichen Verfahren zur Virtualisierung unterstützen und auf unterschiedliche Art und Weise kombinieren. Soll ein DBS virtuell bereitgestellt werden, so sollten die folgenden Anforderungen eingehalten werden:

– Mechanismen für ein automatisches Failover müssen zur Verfügung stehen und das Failover muss für die Applikation und den Anwender transparent sein,

– die Optimierung der Ressourcen muss über den ganzen Server-Park automatisch möglich sein und Lasten müssen einfach und dynamisch verteilt werden können und

– das System muss hoch skalierbar sein.

Neben der Einrichtung virtueller Datenbanken kann natürlich auch das DBS als Anwendung virtualisiert werden.

8.3.1 Servervirtualisierung

Neben der Möglichkeit, das DBS zu virtualisieren, bieten die fortscheitende Entwicklung im Bereich der Servervirtualisierung weitere Ansätze zur Verbesserung der Verfügbarkeit.

Abzuraten ist in jedem Fall von einer kompletten Konsolidierung physikalischer Systeme auf logische Systeme. Für den Einsatz in einer HV-Umgebung müssen in jedem Fall SPoF eliminiert werden, weshalb solche Ansätze nicht tragbar sind.

Vielversprechend können jedoch Varianten sein, in denen virtuelle Systeme auf verschiedene, physikalische Systeme, z. B. in einer Cluster-Konfiguration zum Einsatz kommen. Eine Variante (siehe Abbildung 22) stellt die Errichtung eines Clusters zwischen mehreren virtuellen Maschinen dar, die jeweils auf unterschiedlichen physischen Servern betrieben werden. Durch die Realisierung des Clusters in verschiedenen virtuellen Maschinen, die auf mehrere physische Server verteilt sind, werden SPoF vermieden. Neben Vorteilen im Bereich der Verfügbarkeit kann diese Variante die Sicherungs- und Wiederherstellungsstrategie vereinfachen. So können schnell Snapshots der aktuellen Konfiguration erstellt und im Bedarfsfall verwendet werden.

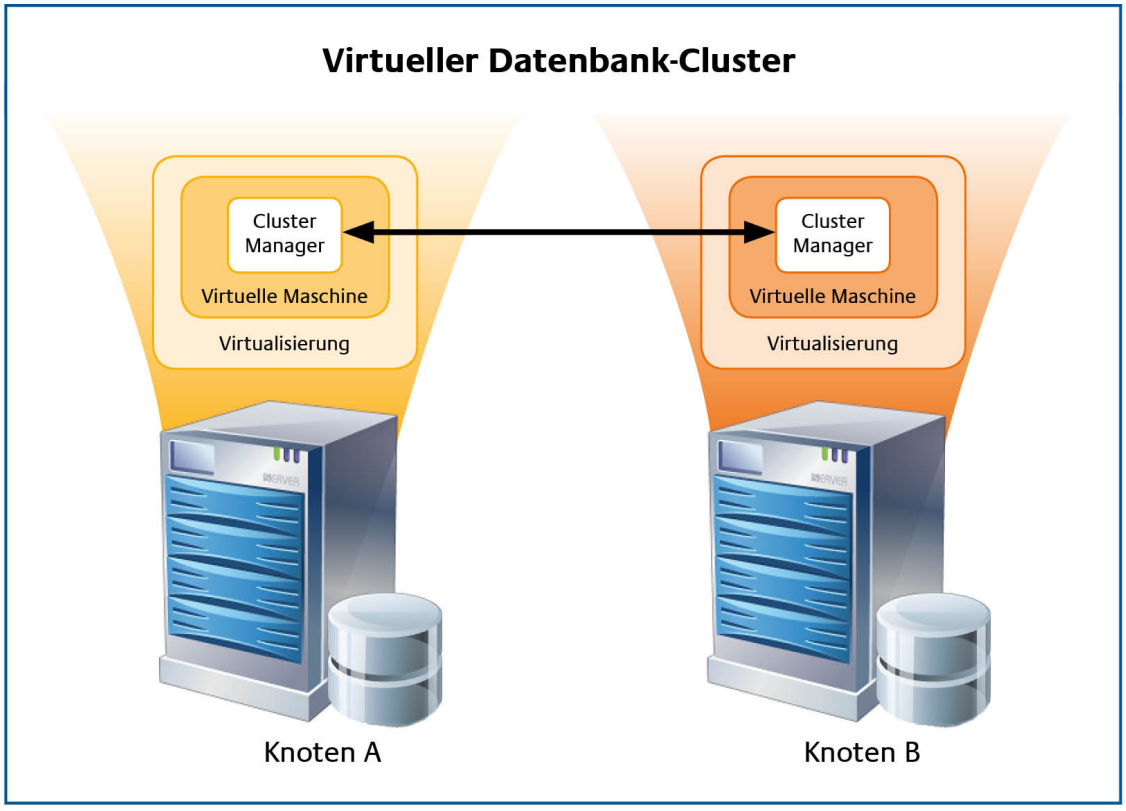


Abbildung 22: Virtualisierung im Cluster

8.4 Betrachtung der Realisierungsalternativen

Der Einsatz von Virtualisierung im HV-Umfeld klingt spannend und vielversprechend. Die Virtualisierung entwickelt sich zügig weiter und erobert immer neue Anwendungsbereiche.

Die Virtualisierung im Bereich der Verzeichnisse und IP-Adressen ist bereits seit Langem gängige Praxis und kann die Verfügbarkeit durch die Reduzierung von SPoF verbessern. Diese Techniken fügen sich transparent in die DBS-Umgebung ein und lassen sich unproblematisch einsetzten.

Diese Aussage kann für den Einsatz von DBS-Virtualisierung sowie der Servervirtualisierung so nicht gelten. Bei der Einführung von Server-Virtualisierungstechniken wird eine zusätzliche Abstraktionsschicht eingeführt die den Komplexitätsgrad weiter erhöht. Mit der Einführung können daher auch ganz neue Probleme auftreten die in einer „dedizierten“ DBS-Architektur nicht auftreten. Es ist daher genau zu prüfen, ob die Vorteile einer Virtualisierung - keine oder weniger redundante Systeme, Clustering mit virtuellen und physikalischen Servern oder virtuelle Cluster, die auf verschiedenen Maschinen verteilt sind – im Rahmen einer HV-DBS einsetzbar sind.

Neben diesen Vorteilen hat die Virtualisierung von Servern/Systemen einen unter Umständen wesentlichen Vorteil. Aufgrund der immer weiter voranschreitenden technischen Entwicklung ist es heutzutage schwierig, nach einigen Monaten ein baugleiches System (Chipsatz, Controller, NIC, usw.) zu erhalten. Gerade komplexe DBS bedürfen einer umfangreichen Konfiguration, die teilweise von der konkreten Hardware abhängig ist. Im Fall eines Hardware-Defektes kann der Hardwareausfall eines Systems zu Problemen führen, wenn kein baugleiches System mehr lieferbar ist. Hier bieten Virtualisierungstechniken durch die Entkopplung der Anwendung von der Hardware Vorteile, da die virtualisierten Instanzen schnell wieder in den Wirkbetrieb überführt werden können. So ermöglicht die Virtualisierung hier eine größere Unabhängigkeit von den Modell-Laufzeiten der Hersteller.

Eine komplette Virtualisierung ganzer DBS-Farmen auf wenige physikalische Systeme ist im HV-Umfeld nicht ratsam. Der HV-konforme Einsatz von DBS macht es erforderlich, weitestgehend auf zusätzliche Abstraktionsebenen zu verzichten, zugunsten optimierter Prozessabläufe des DBMS, welche hohe Verfügbarkeit sicherstellen sollen. So kann der Verzicht auf redundante Systeme zugunsten „virtueller“ Maschinen im Problemfall kritisch werden. Viele Hersteller suggerieren einen geradezu trivialen Umgang mit ihren Virtualisierungslösungen, nach dem Motto: fällt eine Anwendung aus, dann wird „mal eben“ auf eine andere umgeschaltet.

Neben dem Einsatz von virtuellen DBS kann eine partielle Virtualisierung, z. B. auf der Ebene der Verzeichnisstrukturen oder der IP-Adressen, einen wertvollen Beitrag zur Verbesserung der Verfügbarkeit leisten. So kann durch die Nutzung virtueller Verzeichnisse die Ausfallsicherheit eines einzelnen Verzeichnisknotens durch die Verwendung von Aliasnamen verbessert werden.

9 Konzepte für eine hohe Verfügbarkeit

In den folgenden Abschnitten wird anhand konkreter Beispiele gezeigt, in welchen Szenarien die zuvor aufgeführten Prinzipien ihre Anwendung finden. In der Praxis müssen Datenbanken ganz unterschiedliche Anforderungen erfüllen und dementsprechend existiert eine Vielzahl verschiedener Datenbankanwendungen. Trotz der immensen Vielfalt an Anwendungsszenarien lassen sich zwei markante Einsatzszenarien ableiten: zum einen eine Datenbankarchitektur für den überwiegenden Lese-Zugriff mit hohen Anforderungen an die Verfügbarkeit, zum anderen eine eher für Schreibzugriffe optimierte Architektur mit der im B2B- und B2C-Umfeld häufig geforderten Transaktionssicherheit. Das zweite Einsatzszenario erfordert eine Datenbankarchitektur, welche die Transaktionssicherheit über Performace-Anforderungen stellt.

Beide in der Praxis häufig anzutreffenden Szenarien werden in den folgenden Abschnitten dargestellt sowie deren Architektur erläutert.

9.1 Leseoptimierte Abfragearchitektur

Die erste (Beispiel-)Architektur stellt einen hochverfügbaren Verzeichnisdienst dar. Die Hauptaufgabe der Anwendung ist, Anfragen von Clients unmittelbar zu beantworten. Solche Systeme sind leseoptimiert. Antworten sollen innerhalb kürzester Zeit vorliegen und der Dienst muss dauerhaft verfügbar sein. Typisch sind solche Architekturen bei OCSP-Respondern, einer PKI oder Authentifizierungsservern zur Anmeldung von Desktop-PCs. Die Reihenfolge der Beantwortung der einzelnen Anfragen ist prinzipiell bedeutungslos, sofern die Antwortzeit jeder Anfrage innerhalb eines akzeptablen Zeitfensters liegt. Änderungen finden im Vergleich zu den Leseoperationen nur selten statt und können im Zuge des Anfrage-Schedulings mit niedrigerer, aber nicht unbestimmter Priorität bearbeitet werden.

Aufgrund der geringen Interaktion kann ein solches System, wie in der nachfolgenden zu erkennen ist, als Baum-Cluster mit einem Loadbalancer und mehreren Datenbank-Instanzen realisiert werden.

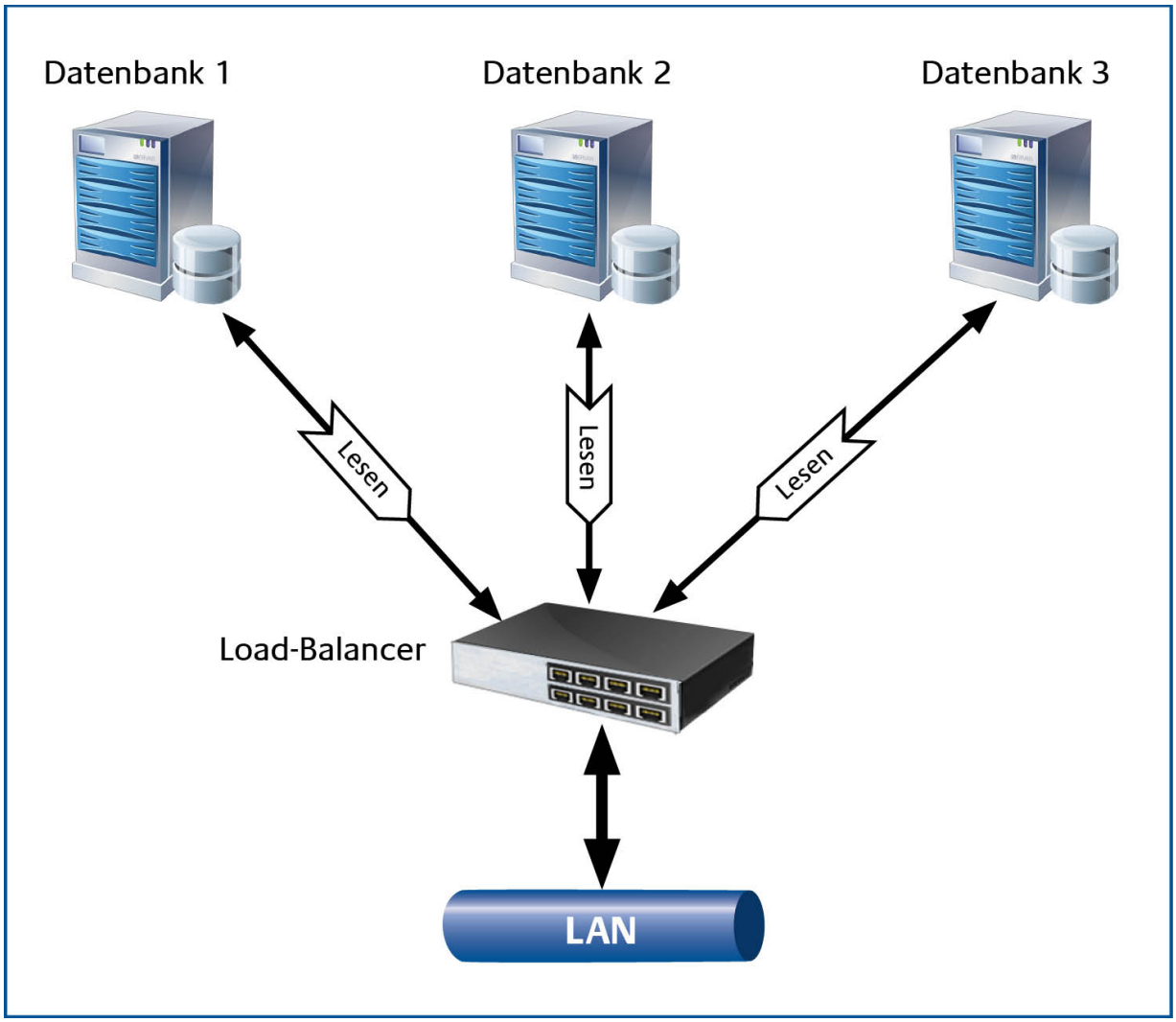


Abbildung 23: Datenbankcluster beim Lesezugriff

Der Zugriff auf das System kann über eine Netzschnittstelle, die auf dem Transport-Protokoll TCP basiert, umgesetzt werden. Der Loadbalancer bestimmt bei jeder neuen Datenverbindung über seine Arbitrationsfunktion eine der aktiven Datenbankinstanzen, die für die komplette Lebensdauer dieser Verbindung zuständig bleiben soll. Als Arbitrationsfunktionen bieten sich dazu an:

- **Round Robin:** Jede Instanz wird der Reihe nach ausgewählt. Wurde jede Instanz einmal berücksichtigt, wird wieder die Erste ausgewählt.
- **Least-Load-Feedback:** Die am wenigsten beschäftigte Instanz erhält die Anfrage.
- **Hash-Based:** Aus den Parametern der Verbindung (etwa der Absender-IP-Adresse) wird über eine Hash-Funktion die zuständige Instanz berechnet.

Solche Szenarien lassen sich etwa für Abfrageprotokolle wie LDAP oder RADIUS einfach realisieren. Auch die herstellerspezifischen SQL-Zugriffsprotokolle können über genormte Schnittstellen (unter anderen ODBC, JDBC oder DBI) mittels Treibern verwendet werden.

Der Schreibzugriff für Änderungen und Ergänzungen des Datenbestandes wird separat gehandhabt. Die Änderungen werden nur auf einer dedizierten Datenbank-Instanz vorgenommen und dort

zwischengespeichert. Wenn die Anfragelast dies erlaubt, wird über einen Replikationsmechanismus der Datenstand zwischen den einzelnen Instanzen abgeglichen. Um einen SPoF für Schreibzugriffe zu vermeiden, ist prinzipiell jede Instanz dazu in der Lage, wobei während des Betriebes nur eine Datenbankinstanz Schreibanfragen entgegen nimmt. Erst wenn alle Daten zwischen den Instanzen ausgetauscht und bestätigt sind, werden diese für die Lesezugriffe frei geschaltet. Dieses Prinzip ist in der dargestellt.

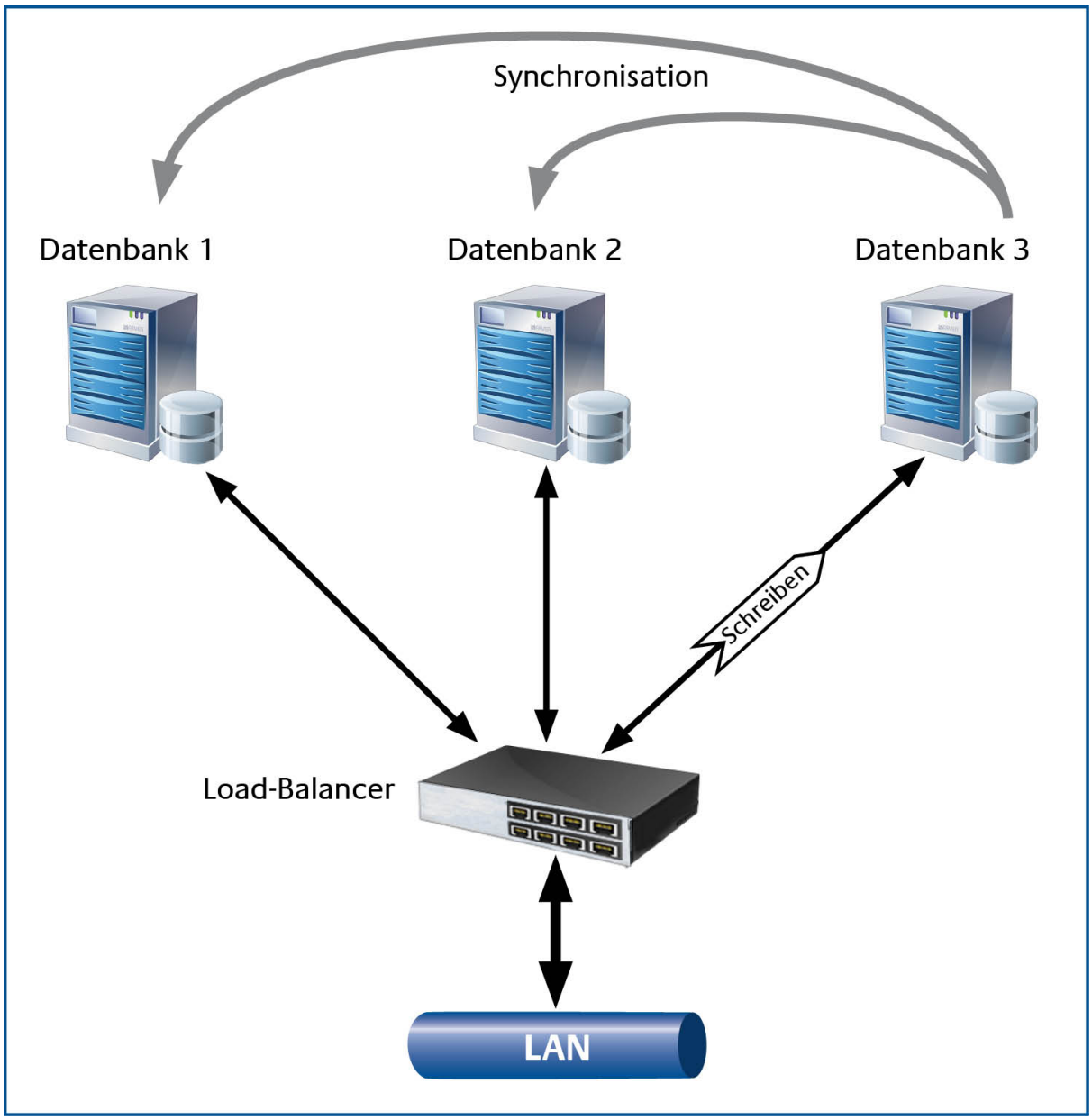


Abbildung 24: Datenbankcluster mit Aktiv/Passiv-Kopplung für Schreibzugriffe

Es liegt somit ein Aktiv/Aktiv-Cluster für den Lese- und eine Aktiv/Passiv-Kopplung für den Schreibzugriff vor.

9.2 Optimierung auf Transaktionssicherheit

Der zweite Anwendungsfall beschreibt eine Architektur, die in Umgebungen zum Einsatz kommt, in welcher der genaue Ablauf von Vorfällen protokolliert werden muss.

Hierzu zählen z. B. Abrechnungs- und Zahlungssysteme oder Authentifizierungsserver. Bei diesen Anwendungen ist die Konsistenz der Daten als vorrangig gegenüber der Performance anzusehen. Es muss in jedem Fall gewährleistet sein, dass selbst bei einem Datenverlust oder dem Ausfall einer Teilkomponente der vollständige Ablauf der Transaktionen nachvollzogen werden kann. Dafür müssen unter hohen Belastungen Verzögerungen bei der Speicherung in Kauf genommen werden.

Das DBMS verarbeitet die Anfragen und sorgt über interne Mechanismen dafür, dass relevante Anfragen an eine Spiegeldatenbank weitergeleitet werden. Eine solche Spiegeldatenbank kann und sollte hinsichtlich ihres Aufstellungsortes und ihrer Betriebsinfrastruktur (Räume, Anbindung an Energie- und Klimakontrolle etc.) möglichst unabhängig vom Primärsystem sein, um neben der Informationsredundanz auch eine geografische Redundanz zu realisieren.

Soll der höchste Grad an Konsistenz gewährleistet sein, so meldet das primäre System den Erfolg einer Transaktion erst dann, wenn es von der Spiegeldatenbank eine Erfolgsmeldung erhalten hat (synchrones Commit). Hierbei sind durchaus auch Caching-Effekte zu berücksichtigen. Je nach Speichermedium kann eine Ebene der Speicherkette bereits eine Vollzugsmeldung (so genannter I/O-Complete) melden, obwohl die Daten sich erst in einem flüchtigen Zwischenspeicher befinden und bei einem Systemabsturz verloren gehen könnten. Daher müssen solche Effekte durch das verwendete System wirkungsvoll ausgeschlossen werden. Die physikalische Speicherung der Daten kann auf dem Datenbanksystem und/oder einem SAN bzw. NAS erfolgen, um zusätzliche Redundanz zu gewährleisten.

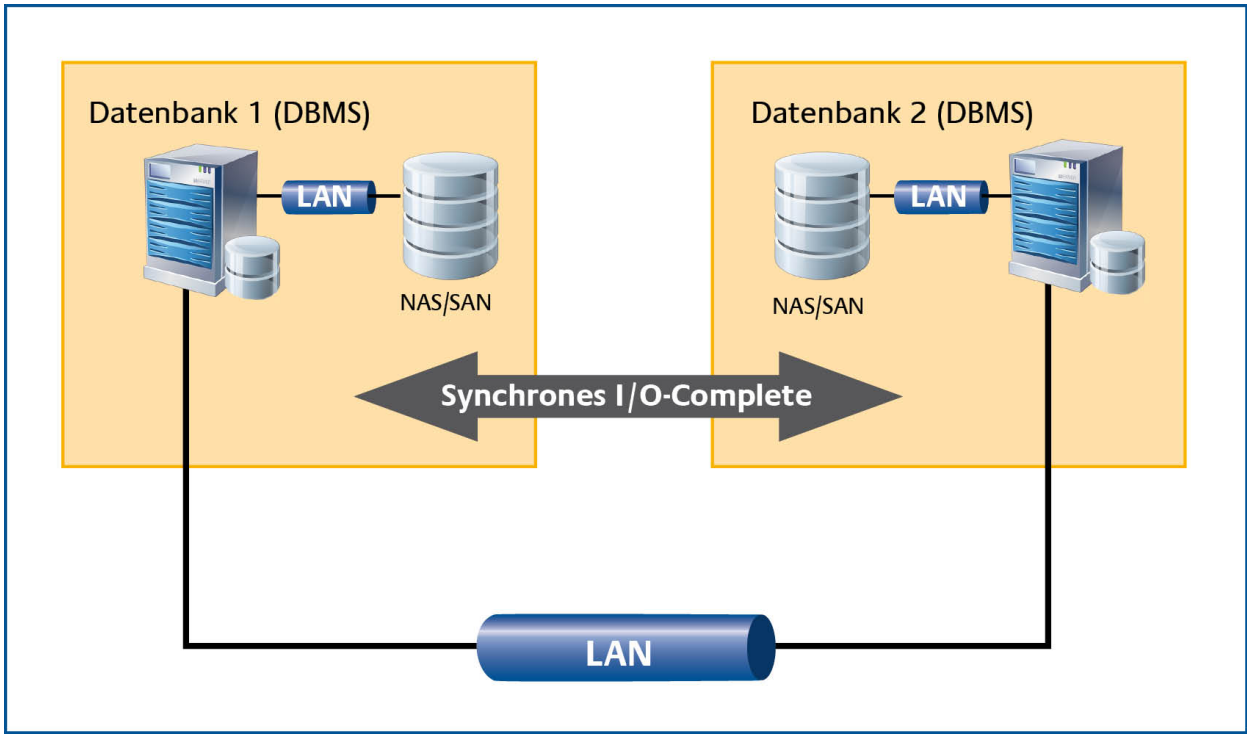


Abbildung 25: Mehrstufiges, transaktionssicheres DBMS

10 Zusammenfassung

Ebenso wie auf allen anderen Betrachtungsebenen gilt auch für die Realisierung hochverfügbarer DBS, dass nur die geeignete Kombination mehrerer Verfügbarkeitsprinzipien zu einer zuverlässig hochverfügbaren DBS-Architektur führt. Mit steigender Komplexität der eingesetzten Komponenten muss neben der Auswahl geeigneter Bausteine (Hardware, Software) auf das reibungslose Zusammenspiel der Komponenten geachtet werden. Darüber hinaus ist ein besonderes Augenmerk auf Architektur- und Designaspekte der IT-Systeme sowie der Infrastruktur zu richten. Zusätzlich wird erkennbar, dass die Auswahl der geeigneten Mechanismen eine sorgfältige Betrachtung und Analyse der Anwendungsszenarien erfordert, da ein HV-Design in der Regel nicht universell eingesetzt werden kann. Der Entwurf muss in jedem Fall auf die Applikation ausgerichtet werden.

So ergibt sich eine Entscheidungspyramide (siehe Abbildung 26), die in Abhängigkeit von den Anforderungen, die sich durch die Geschäftsprozesse ergeben, verschiedene Wege und Verfahren aufzeigt und als Ergebnis eine abgestimmte HV-Strategie darstellt.

Die Entscheidung für den Betrieb eines DBS, welches höchsten Anforderungen an die Verfügbarkeit genügt, führt in der Regel auch zu einer Entscheidung für ein Mehrrechner-System. Nur durch ein Mehrrechnersystem kann der Ausfall eines Knotens kompensiert werden. Die verschiedenen Arten von Mehrrechnersystemen wurden im Abschnitt „Mehrrechner-Datenbanksysteme“ behandelt. Grundsätzlich sind dedizierte Mehrrechnersysteme (Parallelrechner) und Cluster-Systeme für den Einsatz in einer HV-Umgebung zu empfehlen. Bei dem Einsatz in einer Standby-Variante kann typischerweise nur ein Betrieb als Hot-Standby-System den hohen Verfügbarkeitsanforderungen gerecht werden. Das Cold-Standby-Verfahren kann lediglich in Kombination mit einer Cluster-Variante den HV-Anforderungen genügen.

Neben der grundsätzlichen Entscheidung für eine Mehrrechner-Variante kann in HV-Strategien auch der Einsatz von Virtualisierungstechniken mit einbezogen werden. Bei dem Einsatz eines virtuellen DBS oder einer Servervirtualisierung ist jedoch genau zu prüfen, ob die Implementierung keine SPoF aufweist. Ebenso müssen Performancebetrachtungen durchgeführt werden, um zu verhindern, dass im Fehlerfall die erhöhte Last zu unerwünschten Folgeeffekten (Überlast) führt. In eine HV-Strategie müssen die bei Mehrrechnerbetrieb notwendigen Überlegungen zur Wahrung der Transaktionskonsistenz einfließen. Sobald mehr als ein aktiver Knoten Änderungen in der Datenbank zulässt, müssen Mechanismen greifen, die Sicherstellen, dass die Transaktionskonsistenz gewährleistet wird. Hier können, je nach Anforderung, Replikationsverfahren zum Einsatz kommen oder es erfolgt eine asynchrone Aktualisierung durch Verwendung von LOG-Shipping.

Die im Rahmen der eingesetzten Replikationsverfahren getroffene Entscheidung hat nicht nur unmittelbaren Einfluss auf die Performance und Verfügbarkeit des DBS, sondern sie wirkt sich auch mittelbar auf weitere Verfahren aus. So ist die Sicherungs- und Wiederherstellungsstrategie eng mit dem eingesetzten Replikationsverfahren verknüpft. Ebenfalls sind die Überwachung und das Auditing auf die eingesetzte Strategie abzustimmen. Denn nur zuverlässig funktionierende Systemfunktionen, z. B. die Replikation, können eine adäquate Verfügbarkeit sicherstellen. So sollten - im Sinne einer Fehlervermeidungsstrategie - Überwachungs- und Auditfunktionen auf jeder Ebene verankert werden, um auf kritische Änderungen reagieren zu können, bevor Fehler auftreten. Daher stellen die beiden Bereiche - Sicherungs- und Wiederherstellungsstrategie sowie Überwachung und Auditing - wesentliche Säulen bei der Realisierung einer HV-DBS dar und sind

auf die komplette DBS-Architektur anzuwenden. Sie sind daher in der Entscheidungspyramide Ebenen übergreifend dargestellt.

Es zeigt sich, dass für den Entwurf eines HV-DBS ein komplexer Entscheidungsbaum erarbeitet werden muss, der eine Vielzahl von Parametern und Abhängigkeiten auf den verschiedenen Ebenen berücksichtigt. Hierzu zählt unter anderen die Festlegung einer Konfliktstrategie für das Auflösen oder Verhindern von Verklemmungen. Grundsätzlich ist dies bei jedem Mehrrechnersystem, das mehr als einen aktiven Knoten nutzt, zu definieren. Die Wahl des richtigen Sperrverfahrens und des Sperrprotokolls können ebenfalls große Auswirkungen auf die Verfügbarkeit des DBS haben. In der Abbildung 26 ist eine Entscheidungspyramide dargestellt, welche die jeweiligen Entscheidungsalternativen auf den unterschiedlichen Ebenen darstellt. Mit der Realisierung eines HV-DBS ist in typischerweise auch die Entscheidung für eine Mehrrechner-DB gefallen. Auf den folgenden Ebenen sind, in Abhängigkeit der jeweiligen Anforderungen, neue Entscheidungen zu treffen. Z. B. die Entscheidung für eine parallele DB, eine Cluster-DB oder eine Standby-DB. In diese Überlegungen können die Aspekte einer möglichen Virtualisierung mit einbezogen werden. Daran schließen sich Entscheidungen bezüglich der Mehrrechner-Architektur und des DB-Designs an (z. B. Shared Nothing, Aktiv/Aktiv-Cluster oder Standby-Varianten). Abhängig von diesen Entscheidungen, müssen Synchronisations-/Replikationstechnologien oder LOG-Shipping-Verfahren eingesetzt werden, um eine Transaktionskonsistenz zu gewährleisten. Überlegungen hinsichtlich der optimalen Konfliktstrategie sind ebenfalls erforderlich. Alle Entscheidungen bauen aufeinander auf und die zum Einsatz kommende Sicherungs- und Überwachungsstrategie ist mit den gewählten Verfahren abzustimmen.

Die Motivation für den Aufbau eines HV-DBS ergibt sich aus den Anforderungen der Geschäftsprozesse. Für die entsprechende Umsetzung ist eine geeignete HV-Strategie erforderlich. Der vorliegende Beitrag bietet bezogen auf Datenbanksysteme Hilfestellung beim Entwurf entsprechender Strategien. Die geeigneten Lösungsansätze sind immer im Kontext der Gesamtarchitektur zu sehen. Im fachlichen Zusammenhang zu den hier erwähnten Ansätzen stehen insbesondere folgende Beiträge des HV-Kompendiums:

- Netzwerk
- Cluster-Architekturen
- Server (Hardware)
- Speichertechnologien
- Überwachung

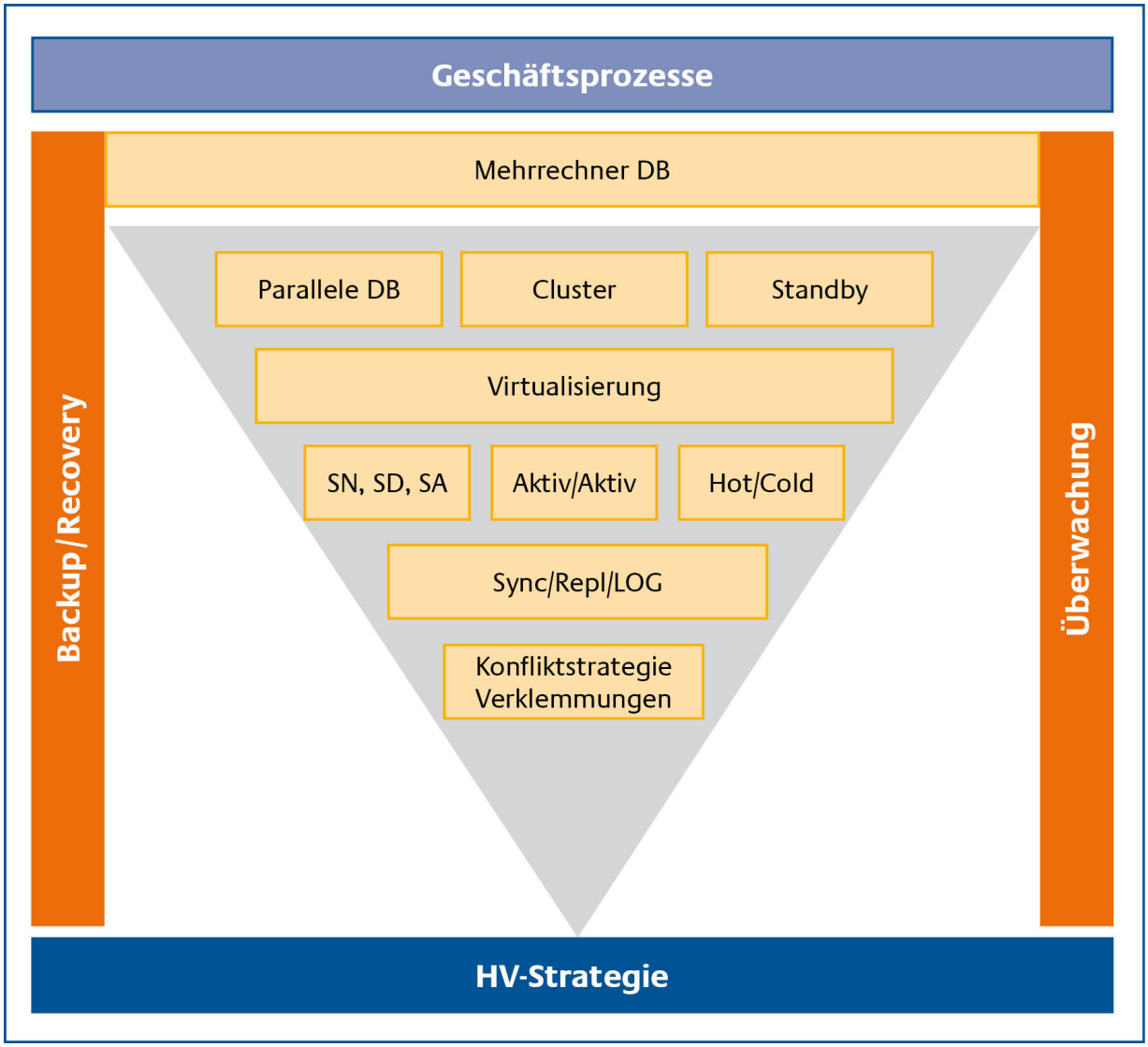


Abbildung 26: Entscheidungspyramide

Anhang: Verzeichnisse

Abkürzungsverzeichnis

Ein komplettes Verzeichnis hierzu findet sich in Band AH, Kapitel 5

Glossar

Ein komplettes Verzeichnis hierzu findet sich in Band AH, Kapitel 6

Literaturverzeichnis

Ein komplettes Verzeichnis hierzu findet sich in Band AH, Kapitel 7