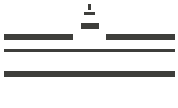

Hauptspeicherdatenbanken — Denkgeschwindigkeit auch für KMU?

Abschlussbericht zum Projekt „Das Potential von In-Memory-Datenbanken in und für KMU“

Jens Lechtenbörger, Vanessa Jie Ling, Gottfried Vossen
Institut für Wirtschaftsinformatik
Westfälische Wilhelms-Universität Münster

April 2015

Inhaltsverzeichnis

Executive Summary

Dieser Bericht stellt Einsatzmöglichkeiten, State-of-the-Art sowie mögliche Herausforderungen bei der Einführung von Hauptspeicherdatenbanktechnologie herstellerneutral dar. Es werden Anwendungsbeispiele für die Nutzung von Hauptspeicherdatenbanken skizziert, wobei sowohl generelle Überlegungen hinsichtlich der zu erwartenden Vorteile angestellt als auch konkrete Unternehmensprojekte vorgestellt werden.

Auf technischer Ebene werden die zentralen Grundlagen zusammengetragen, die für das Verständnis der im Zusammenhang mit Hauptspeicherdatenbanken beworbenen Geschwindigkeitsvorteile notwendig sind. Aufbauend auf diesen Grundlagen werden ausgewählte Datenbanksysteme mit ihren Besonderheiten präsentiert, wobei neben bekannten kommerziellen Systemen auch wegweisende universitäre Entwicklungen einbezogen werden. Im Anschluss werden mögliche Herausforderungen bei der Einführung von Hauptspeicherdatenbanken diskutiert, bevor der Bericht mit einem Ausblick endet.

1. Einleitung

„Während des vergangenen Jahrzehnts sind Hauptspeicherpreise rapide gefallen, und sie werden dies voraussichtlich weiterhin tun. [...] Mit der Verfügbarkeit größerer Hauptspeicher wird es möglich, die Speicherung von Datenbanken im Hauptspeicher in Erwägung zu ziehen.“ Diese Sätze könnten der aktuellen Fachpresse entnommen sein, tatsächlich entstammen sie einer mehr als 30 Jahre alten Publikation von DeWitt et al. [DKO+84] aus dem Jahre 1984 (als Preise von 1.500 US$ pro *Megabyte* Hauptspeicher als rapide gefallen angesehen wurden). Im selben Jahr stellten Garcia-Molina et al. [GLV84] eine Architektur mit „massivem Hauptspeicher“ für speicherintensive Berechnungen mit höchsten Leistungsansprüchen vor, und 1989 erschien bereits ein Übersichtsartikel zu Hauptspeicherdatenbanken (engl. *in-memory* oder *main-memory databases*) [Eic89], der die hierarchische Datenbank IMS/VS Fast Path von IBM als seit den 1970er Jahren kommerziell verfügbare Hauptspeicherdatenbank nennt und unter anderem feststellt, dass die Frage nach Hauptspeicherdatenbanken nicht mehr „Warum?“ sondern „Warum nicht?“ lauten sollte. Nach dieser anfänglichen Euphorie verschwand das Thema in den folgenden Jahren jedoch aus dem öffentlichen sowie dem wissenschaftlichen Fokus.

Eine Datenbank gilt [Eic89] folgend als *Hauptspeicherdatenbank*, wenn sich der primäre Datenbestand im Hauptspeicher befindet; Festplatten werden lediglich für Backup und Recovery verwendet. Der vorliegende Bericht ist zudem auf *relationale*, den Sprachstandard SQL unterstützende Datenbanken (für eine Einführung siehe z. B. [Vos08]) fokussiert, welche die bei weitem populärste Form von Datenbanken darstellen.

Trotz der Forschungsaktivitäten der 1980er Jahre gewinnen Hauptspeicherdatenbanken erst seit wenigen Jahren wieder zunehmend an Popularität, wenn es um höchste Geschwindigkeitsansprüche geht. In der Tat gibt es gute Gründe für die Entwicklung spezialisierter Datenbanksysteme, denn in [HAMS08] wurde gezeigt, dass traditionelle, festplattenbasierte Datenbanksysteme im populären TPC-C-Benchmark zur Leistungsmessung der Transaktionsverarbeitung, selbst wenn alle Daten vollständig in den Hauptspeicher passen, nur 10% der Zeit mit echter Transaktionsverarbeitung verbringen, während die verbleibenden 90% auf vier Quellen von Mehraufwand entfallen, nämlich die Verwaltung des Buffer-Pools, Verwaltung von Multi-Threading, Sperrverfahren auf Zeilenebene und Write-Ahead-Logging. Entsprechend liegt es nahe, spezialisierte Hauptspeicherdatenbanksysteme zu entwickeln, in denen diese Arten von Mehraufwand eliminiert oder zumindest reduziert werden.

Mittlerweile bieten auch die großen Datenbankhersteller (insbesondere IBM, Microsoft und Oracle) hauptspeicherbasierte Produkte an und bewerben diese mit Geschwindigkeits-

verbesserungen um Faktoren bis zu 100.000. Damit könnten Analysen oder Prognosen, deren Berechnungen bisher Tage dauerten, nun innerhalb von Sekunden durchgeführt werden, was neuartige Anwendungen ermöglicht. Vor diesem Hintergrund sprechen Plattner und Zeier [PZ12] von einer Revolution für die Entwicklung betrieblicher Anwendungen. Während in dieser Diskussion vor allem große Unternehmen mit Hauptspeicherdatenbanken in Verbindung gebracht werden, sind die Anwendungsmöglichkeiten für kleine und mittlere Unternehmen (KMU), die laut EU-Definition[1] einen Umsatz von bis zu 50 Mio. € bei weniger als 250 Mitarbeitern aufweisen, noch deutlich weniger klar.

Entsprechend strebt dieser Bericht an, Einsatzmöglichkeiten, State-of-the-Art sowie mögliche Herausforderungen bei der Einführung von Hauptspeicherdatenbanktechnologie herstellerneutral darzustellen. Mit diesem Ziel werden in Abschnitt 2 Anwendungsbeispiele für die Nutzung von Hauptspeicherdatenbanken skizziert, wobei sowohl generelle Überlegungen hinsichtlich der zu erwartenden Vorteile angestellt als auch konkrete Unternehmensprojekte vorgestellt werden. Technische Grundlagen, die für das Verständnis der im Zusammenhang mit Hauptspeicherdatenbanken beworbenen Geschwindigkeitsvorteile notwendig sind, werden in Abschnitt 3 zusammengetragen, bevor in Abschnitt 4 ausgewählte Datenbanksysteme mit ihren Besonderheiten präsentiert werden, wobei neben bekannten kommerziellen Systemen auch wegweisende akademische Systeme einbezogen werden. Abschnitt 5 ist dann der Diskussion möglicher Herausforderungen bei der Einführung von Hauptspeicherdatenbanken gewidmet, bevor der Bericht in Abschnitt 6 zusammengefasst und mit einem Ausblick beendet wird.

[1] `http://ec.europa.eu/enterprise/policies/sme/facts-figures-analysis/sme-definition/index_de.htm`

2. Anwendungsbeispiele

Hauptspeicherdatenbanken können aus Gründen, die wir im folgenden Abschnitt diskutieren werden, enorme Geschwindigkeitsvorteile gegenüber traditionellen, festplattenbasierten Datenbanken ausspielen. Im Prinzip lässt sich eine festplattenbasierte Datenbank, die im Hintergrund einer „zu langsamen“ Anwendung den Flaschenhals bildet, „einfach“ durch eine Hauptspeicherdatenbank austauschen, jedenfalls wenn sichergestellt ist, dass die Anwendung keinen herstellerspezifischen SQL-Dialekt voraussetzt, sondern eine Teilmenge des SQL-Standards nutzt, die von beiden Datenbanken verstanden wird. So berichtete Oracle[2] zur Vorstellung seiner neuen, hauptspeicherfähigen Datenbankversion 12c von Beschleunigungsfaktoren bis zu 3.500 für die hauseigene Unternehmenssoftware (Peoplesoft, JD Edwards), die in vielen Fällen zuvor langwierige Berechnungen zu interaktiven Vorgängen machen.

Losgelöst von konkreten Anwendungsszenarien hat der Geschwindigkeitsvorteil von Hauptspeicherdatenbanken mehrere allgemeine Implikationen. Wir unterscheiden im Folgenden drei Arten von Vorteilen, die sich unterschiedlichen Anforderungen an Datenverarbeitung zuordnen lassen. Erstens verspricht die Hauptspeicherverarbeitung generelle Geschwindigkeitssteigerungen. Zweitens können diese Geschwindigkeitsvorteile den Schlüssel zur Erschließung neuartiger (Big-Data-) Szenarien darstellen, wo Daten in bisher unzugänglichem Umfang und Detailgrad vorliegen oder produziert werden. Drittens können IT-Landschaften vereinfacht werden, wenn Systeme zusammengeführt werden, die zuvor aufgrund von Leistungsgrenzen getrennt betrieben wurden.

2.1. Geschwindigkeitssteigerung

Plattner und Zeier sprechen in [PZ12] plastisch von der Denkgeschwindigkeit (engl. *speed of thought*), die als Antwortzeit von etwa 550–750*ms* von jeder Anwendung erreicht werden sollte, um flüssiges und effektives Arbeiten zu ermöglichen: Falls die Antwort auf eine Nutzerinteraktion länger dauert als diese Denkgeschwindigkeit, wird die resultierende Pause als Wartezeit wahrgenommen, und das Gehirn schweift von der eigentlichen Aufgabe ab zu anderen Dingen. Entsprechend ist dann kein konzentriertes Arbeiten mehr möglich. Durch den Einsatz von Hauptspeicherdatenbanken ist zu erwarten, dass vermehrt Anwendungen diese Denkgeschwindigkeit erreichen können.

Beispielsweise berichtet Oracle[3] mit Bezug auf die ERP-Software JD Edwards von Analysen auf Bestellungen mit 104 Mio. Bestellpositionen. Hier konnten Datenbankanfragen

[2] `http://www.oracle.com/us/corporate/events/dbim/index.html`
[3] `http://www.oracle.com/us/corporate/events/dbim/index.html`

zur Beantwortung unvorhergesehener Rückfragen von Kunden um einen Faktor von 1.700 beschleunigt werden, von zuvor 22,5 Minuten auf Sekundenbruchteile. Der Vorteil für die Interaktion mit Kunden liegt auf der Hand.

Natürlich führt nicht jede Geschwindigkeitssteigerung direkt zur Denkgeschwindigkeit. Dennoch können Reduktionen der Berechnungszeiten von Stunden auf Minuten oder Sekunden zu deutlichen Geschäftsvorteilen führen. Als erstes Beispiel sei die Immonet GmbH (280 Mitarbeiter) genannt, ein Internet-Immobilienportal, das seine Auswertungen zu Markttrends, zu Kaufmustern von Kunden und zum Maklerverhalten auf die Exalytics In-Memory Machine X2-4 von Oracle umgestellt hat.[4] Generell wurden dramatische Verbesserungen in der Ausführung von Abfragen und Erstellung von Berichten erzielt, von Tagen oder Stunden auf Minuten oder Sekunden. In der Folge konnte das Unternehmen seine Suchmaschinenwerbung verbessern und die Kundenanfragen um 300% und die Umsatzerlöse um 200% steigern.

Ein ähnlicher Fall liegt bei der Balluff GmbH (2.750 Mitarbeiter) vor, wo SAP-Software inklusive SAP Business Warehouse auf IBM-Hardware mit der zugrunde liegenden Datenbank IBM DB2 betrieben wurde. Das Unternehmen verzeichnete im Zuge der Migration zu DB2 mit BLU Acceleration deutliche Beschleunigungen bei der Berechnung von Berichten (im Durchschnitt von 30% mit bis zu 98% bei einzelnen Berichten).[5]

Als den SAP Customer Testimonials[6] entstammendes Beispiel für den Einfluss beschleunigter Berechnungen bei produzierenden Unternehmen sei die Royal Swaziland Sugar Corporation (3.500 Mitarbeiter) genannt, ein Zucker- und Ethanolproduzent. Dort wurde SAP HANA zur Verbesserung der Ressourcenplanung eingeführt, und die Dauer des Planungsprozesses konnte von zwei Wochen auf wenige Stunden reduziert werden. Im Ergebnis wird von täglichen Einsparungen von € 75.794 durch gesenkten Zucker- und Wasserverbrauch berichtet.

Schließlich wurde bei bwin, einem führenden Anbieter von Online-Gaming-Angeboten, der SQL Server von Microsoft, der im Backend der Web-Server eingesetzt wurde, zu einem Flaschenhals, der nicht mehr als 15.000 Anfragen pro Sekunde verarbeiten konnte. Durch die Umstellung auf die Hauptspeichertechnologie Hekaton werden nun 250.000 Anfragen pro Sekunde ermöglicht, was den bisherigen Flaschenhals eliminiert hat.[7]

[4] http://www.oracle.com/us/corporate/customers/customersearch/immonet-exalytics-ss-1955430-de.html

[5] http://public.dhe.ibm.com/common/ssi/ecm/sp/en/spc03502wwen/SPC03502WWEN.PDF

[6] http://www.sap.com/customer-testimonials/index.html

[7] http://www.microsoft.com/en-us/showcase/details.aspx?uuid=9585e3c1-2bd2-4559-8aa8-11fe698c5189

2.2. Neue Anwendungen, Big-Data-Verarbeitung

Die Geschwindigkeit der Hauptspeicherverarbeitung ermöglicht neuartige Anwendungen, die auf der Analyse zuvor unzugänglicher Datenbestände beruhen. Zum einen können größere Datenvolumina in kürzerer Zeit ausgewertet werden, zum anderen lassen sich nun auch Datenströme erfassen und auswerten, deren Geschwindigkeit zu hoch für die Verarbeitung in festplattenbasierten Systemen ist. Entsprechend werden Hauptspeicherdatenbanken häufig mit Big-Data-Szenarien in Verbindung gebracht, die durch verschiedene „Vs“ charakterisiert werden (vgl. [Vos14]) wie Volumen oder Größe (eng. *volume*), (hohe) Geschwindigkeit (engl. *velocity*), (hohe) Vielfalt (engl. *variety*), (nicht immer gegebene) Präzision, Genauigkeit oder Vertrauenswürdigkeit (engl. *veracity*) und Wert (engl. *value*).

Derartige Anwendungsmöglichkeiten entstehen beispielsweise rund um die Auswertung von und Reaktion auf Sensordaten (bzw. allgemeiner durch die Behandlung von Datenströmen): Leistungsfähigere Analysesysteme ermöglichen die Auswertung von und Reaktion auf feinergranularen Daten in kürzeren Intervallen. So steht laut SAP Customer Testimonials[8] beim Fußballverein TSG Hoffenheim die Auswertung von Sensordaten in Echtzeit im Fokus, um Trainingsstände und Leistungen der Spieler zu verbessern. Zu diesem Zweck sind sowohl die Bälle als auch die Schienbeinschützer mit Chips ausgestattet, die innerhalb von 10 Minuten bei 16 Spielern mit 6 Bällen knapp 13 Mio. Datenpunkte generieren, welche die Bewegungen von Spielern und Bällen repräsentieren. Diese Daten werden in SAP HANA in Echtzeit verarbeitet und zur Erstellung personalisierter Trainingspläne, für umfassendere Spielanalysen, zur Erprobung von Spielstrategien und im Rahmen der Talentsuche genutzt.

Klassisch betriebliche Szenarien zur Auswertung von Sensordaten finden sich beispielsweise im Kontext des intelligenten Stromnetzes (engl. *smart grid*). Während Abrechnungen und Analysen zum Stromverbrauch traditionell über größere Abrechnungsperioden auf hohem Aggregationsniveau per Batch-Prozess behandelt wurden, besteht im Rahmen intelligenter Stromnetze der Bedarf, Stromproduktion, -speicherung und -verbrauch in Echtzeit zu steuern. In diesem Kontext wird in [SKZP10] ein Prototyp basierend auf einer Hauptspeicherdatenbank vorgestellt, der die detaillierten Verbrauchsdaten ohne Voraggregation erfasst und Einzelkunden betreffende Auswertungen innerhalb von Sekundenbruchteilen erlaubt, was die Entwicklung neuer Abrechnungsmodelle möglich macht.

Ein anderer in den SAP Customer Testimonials[9] genannter Anwendungshintergrund besteht beim Deutschen Fußball-Bund (DFB, 220 Mitarbeiter). Der DFB betreibt ein

[8] `http://www.sap.com/customer-testimonials/index.html`
[9] `http://www.sap.com/customer-testimonials/index.html`

CRM-System auf der Basis von SAP HANA Enterprise Cloud, wodurch eine Ticketing-Lösung realisiert wird und personalisierte Marketing-Kampagnen unter Kontrolle des DFB durchgeführt und ausgewertet werden können.

2.3. Vereinfachung von IT-Landschaften

Der Einsatz von Hauptspeicherdatenbanken verspricht die Vereinfachung von IT-Landschaften, in denen bisher aus Leistungsgründen verschiedene Datenbank-Server für operative und dispositive Anwendungen eingesetzt werden. Dies trifft insbesondere auf die klassische Trennung von OLTP (OnLine Transaction Processing) zur Unterstützung von Geschäftsprozessen im Tagesgeschäft und Business-Intelligence-Anwendungen wie OLAP (OnLine Analytical Processing) zu Auswertungszwecken zu. Aufgrund der gesteigerten Leistungsfähigkeit von Hauptspeicherdatenbanken lassen sich auch komplexe OLAP-Auswertungen direkt auf operativen Detaildaten durchführen, ohne dass Optimierungstechniken wie Voraggregation nötig wären und ohne operative Anwendungen auszubremsen. Neue Akronyme, die diese Kombination von OLTP und OLAP vor allem in Marketing-Materialien verdeutlichen, sind OLXP, OLTAP oder HTAP (Hybrid Transactional/Analytical Processing).

Das in den SAP Customer Testimonials[10] genannte Unternehmen neckermann.at GmbH (100 Mitarbeiter) nutzt die Hauptspeicherdatenbank im Sinne der oben angesprochenen „einfachen“ Beschleunigung einer bestehenden Anwendung. Konkret wurde das ERP-System von der bisherigen Datenbank auf SAP HANA migriert, und zwar im Rechenzentrum eines IT-Providers, wodurch zum einen erhebliche Leistungsverbesserungen für Finanz- und Controlling-Prozesse erzielt wurden und zum anderen die eigene IT-Landschaft vereinfacht werden konnte. Zudem ist es nun möglich, Vorhersagen für Änderungen im Kundenverhalten in Echtzeit zu berechnen.

In ähnlicher Weise wird bei Schaidt Innovations GmbH & Co. KG (530 Mitarbeiter) das ERP-System ebenfalls auf SAP HANA betrieben, hier allerdings als Cloud-Lösung, worin das Unternehmen eine skalierbare IT-Infrastruktur sieht, die das geplante Unternehmenswachstum unterstützen soll.

[10] `http://www.sap.com/customer-testimonials/index.html`

3. Technische Grundlagen

Die Geschwindigkeitsvorteile von Hauptspeicherdatenbanken gegenüber traditionellen, festplattenbasierten Datenbanken basieren hauptsächlich auf der einfachen Idee, die vergleichsweise langsame Mechanik von Festplatten zu eliminieren, indem benötigte Daten komplett im Hauptspeicher vorgehalten werden. Wie wir in der Einleitung erwähnt haben, ist allerdings zu bedenken, dass es *nicht* damit getan ist, eine „normale“ Datenbank mit großzügig dimensioniertem Hauptspeicher auszustatten und auf höchste Leistung zu hoffen: Harizopoulos et al. [HAMS08] haben gezeigt, dass neuartige Techniken nötig sind, um verschiedene Arten von Mehraufwand zu vermeiden, aber gleichzeitig die bewährten ACID-Garantieen der Transaktionsverarbeitung zu gewährleisten. In den folgenden Unterabschnitten erläutern wir die technischen Grundlagen, die sich zur praktischen Umsetzung der einfachen Idee von Hauptspeicherdatenbanken als essentiell herausgestellt haben.

3.1. Speicherhierarchie

In modernen Rechnern stehen verschiedene Arten von flüchtigen und nichtflüchtigen Speicherkomponenten zur Verfügung, die sich in ihrer Größe, ihrer Geschwindigkeit (Latenz und Bandbreite) und ihrem Preis unterscheiden und die typischerweise in der in Abbildung 1 gezeigten Hierarchie angeordnet werden. In dieser Hierarchie nimmt von oben nach unten die Größe des verfügbaren Speichers zu, während Preis und Geschwindigkeit sinken.

Am oberen Ende finden sich die auf oder nahe dem Prozessor untergebrachten, schnellsten und teuersten, aber kleinen Komponenten Prozessorregister und Prozessor-Cache mit Zugriffszeiten im Nanosekundenbereich. Ebenfalls noch auf dem Mainboard untergebracht ist der in diesem Bericht im Vordergrund stehende Hauptspeicher mit Zugriffszeiten von ca. $100ns$. All diese Speicherkomponenten sind *flüchtig*, verlieren also ihre Inhalte bei Stromunterbrechungen, was bei den *nicht-flüchtigen* (auch *stabil* oder *persistenten* genannten) Sekundärspeichern der unteren Ebenen nicht der Fall ist. Letztere lassen sich zudem noch in ständig angeschlossene Online-Speicher wie Flash-Bausteine (SSDs, wahlfreier Zugriff bei etwa $150\mu s$) und Festplatten (Suchzeiten von $10ms$) sowie bei Bedarf gewechselten Offline-Medien wie Magnetbänder unterscheiden.

Um die Latenzen niedrigerer Ebenen zu eliminieren, werden in Hauptspeicherdatenbanken beim Start alle benötigten Daten in den Hauptspeicher geladen, so dass Anfragen ohne ausbremsende Zugriffe auf nicht-flüchtige Sekundärspeicher durchgeführt werden können. Ganz ohne nicht-flüchtigen Speicher kommen Hauptspeicherdatenbanken natürlich

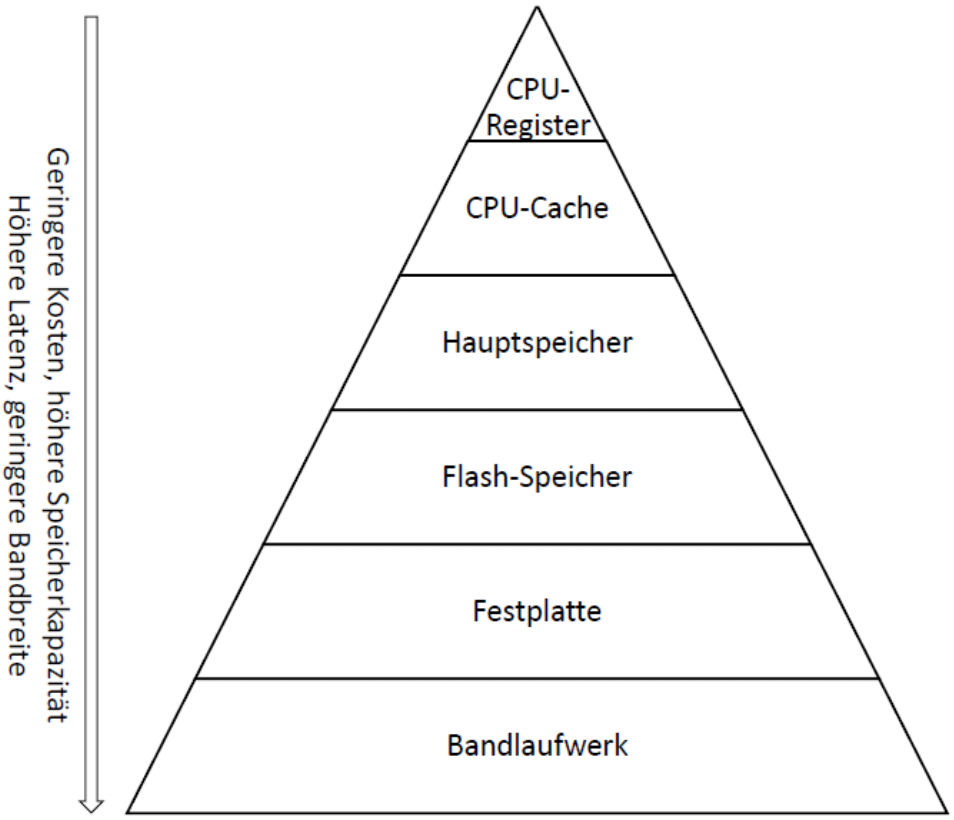

Abbildung 1: Die Speicherhierarchie.

trotzdem nicht aus, da auch sie die von relationalen Datenbanken erwarteten *transaktionalen ACID-Garantieen* Atomarität (engl. *atomarity*), Konsistenz (engl. *consistency*), Isolation (engl. *isolation*) und Dauerhaftigkeit bzw. Persistenz (engl. *durability*) geben (vgl. [Vos08]). Um diese Garantieen auch angesichts nicht vermeidbarer Fehler und Ausfälle durchzusetzen, bedarf es geeigneter Log-Daten im nicht-flüchtigen Speicher, auf denen dann Transaktionskontroll- und Wiederherstellungsmechanismen aufsetzen können. Die zu diesem Zweck benötigten Log-Daten sind allerdings um Größenordnungen kleiner als die eigentliche Datenbank.

Für einen Datenbestand gegebener Größe lässt sich nach einer einfachen Faustformel abschätzen, wieviele CPUs und Server zu seiner Verarbeitung im Hauptspeicher mit Antwortzeiten unterhalb einer Sekunde benötigt werden: Ein einzelner CPU-Kern kann laut [Pla14a] etwa 4 MB im Hauptspeicher vorliegende Daten pro Millisekunde mit Vergleichs- oder Filteroperationen verarbeiten, also 4 GB pro Sekunde, was bei einer 15-Kern-CPU (wie Intels Ivy Bridge EX) 60 GB/*s* und für einen Server mit 8 derartigen CPUs 480 GB/*s* ergibt. Für Server-Cluster mit 10 Servern dürfte es dann schwierig werden, Geschäftsanwendungen zu finden, deren Berechnungen sich nicht innerhalb einer Sekunde durchführen lassen.

Zu bedenken ist bei derartigen Kalkulationen, dass die Größe des im Hauptspeicher vorzuhaltenden Datenbestandes durch Kompressions- und Partitionierungstechniken reduziert werden kann, wie in den folgenden Unterabschnitten erläutert wird.

Schließlich sei angemerkt, dass die Geschwindigkeit von CPUs in den vergangenen Jahren deutlich stärker zugenommen hat als die des Hauptspeichers. Dadurch kann der Hauptspeicherzugriff selbst zu einem Flaschenhals werden, was als Speicherwand (engl. *memory wall*) bezeichnet wird. Um dieser Speicherwand Rechnung zu tragen, sind zahlreiche Anfrageverarbeitungstechniken und zugehörige Datenstrukturen entwickelt worden, die die CPU-Caches durch eine sorgfältige Planung der Speicherzugriffsmuster möglichst gut ausnutzen. Die Kernidee derartiger Planungen besteht darin, wahlfreie Datenzugriffsmuster auf Bereiche zu beschränken, die in den CPU-Cache passen und dadurch Cache-Zugriffsfehler vermeiden.

3.2. Spaltenbasierte Datenorganisation und Kompression

Die Mehrzahl relationaler Datenbanken speicherte Datensätze (Tupel) jahrzehntelang zeilenorientiert. Für jede Zeile einer Tabelle wurden die Attributwerte der Zeile hintereinander auf Festplatte gespeichert. Ein Scan einer Tabelle erforderte dann das Einlesen *aller* Attribute, unabhängig von der Notwendigkeit einzelner Attribute im Kontext einer Anfrage.

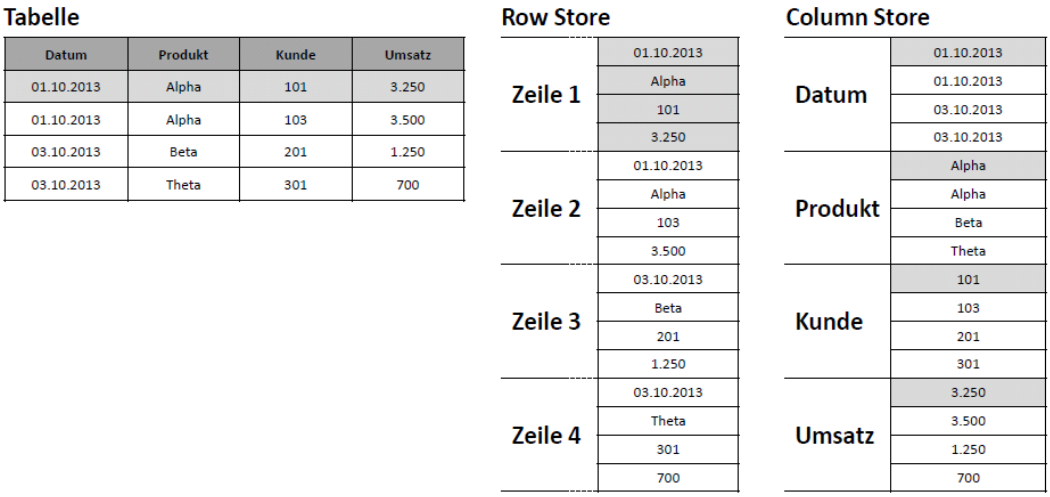

Abbildung 2: Zeilen- und spaltenorientierte Datenorganisation.

Während diese physische Datenorganisation im Rahmen operativer Informationssysteme das OLTP mit Lese-, Einfüge- und Änderungsoperationen auf *einzelnen* Datensätzen performant unterstützen kann, entsteht für Business-Intelligence-Anwendungen wie OLAP mit Aggregationen über viele Zeilen auf ausgewählten Attributen ein nicht zu unterschätzender Mehraufwand.

Dieser Mehraufwand kann durch eine spaltenorientierte Datenorganisation in sog. *Column Stores* vermieden werden, wo die einzelnen Attributwerte einer Spalte hintereinander gespeichert werden, typischerweise in einheitlicher Reihenfolge für alle Spalten einer Tabelle, wie in Abbildung 2 illustriert wird. Das Lesen und Einfügen eines vollständigen Datensatzes erfordern dann relativ aufwändige Zugriffe auf sämtliche, unabhängig gespeicherten Spalten, während im Zuge der Anfrageverarbeitung nur noch wirklich benötigte Attribute in den Hauptspeicher transferiert werden müssen.

Zudem hat sich gezeigt, dass die Werte einzelner Spalten oftmals so homogen sind, dass auch leichtgewichtige Kompressionsverfahren wie Dictionary-Encoding oder Run-Length-Encoding zu signifikanten Kompressionsraten führen (vgl. [PZ12] für konkrete Zahlenbeispiele), wodurch weniger Daten in den Hauptspeicher geladen werden müssen und der zur Verfügung stehende Hauptspeicher effizienter genutzt werden kann. Entsprechend gehört die komprimierte Datenhaltung gepaart mit Anfrageverarbeitungstechniken, die auf komprimierten Daten (also ohne vorherige Dekompression) operieren können, heute zum Standardrepertoire von spaltenorientierten Datenbanken.

Der Vollständigkeit halber sei erwähnt, dass der oben angesprochene Aufwand beim Einfügen von Daten und bei der Rekonstruktion vollständiger Datensätze durch geeignete Techniken reduziert werden kann. So verwaltet beispielsweise SAP HANA einen speziellen

Schreib- oder Deltapuffer, in dem neue Datensätze zeilenorientiert aufgenommen werden, bevor sie (periodisch oder bei Erreichen einer Schwellgröße) in den spaltenorientierten Speicher integriert werden. Anfrage- und Transaktionsverarbeitungstechniken werden dann angepasst, um auf den aus spaltenorientierten Daten und Pufferspeicher bestehenden, konsistenten Datenbestand zuzugreifen. Zum anderen kann in Zwischenschritten der Anfrageverarbeitung oftmals auf die Konstruktion vollständiger Tupel verzichtet werden, wenn *späte* Materialisierungstechniken (engl. *late materialization*) eingesetzt werden. Für Details derartiger Techniken verweisen wir auf [Pla14a].

Insgesamt sind die Vorteile der spaltenorientierten Datenhaltung so groß, dass viele relationale Datenbanken sie heutzutage zumindest als Option anbieten, während sie in Hauptspeicherdatenbanken den Regelfall darstellt.

3.3. Partitionierung

Auch wenn verfügbare Hauptspeichergrößen weiter steigen und zu verarbeitende Datenvolumina in Hauptspeicherdatenbanken dank der im vorangehenden Abschnitt angesprochenen Kompressionstechniken reduziert werden können, passen oftmals nur Teile des Datenbestandes in den Hauptspeicher. In solchen Situationen können zwei nachfolgend skizzierte Ausprägungen horizontaler Partitionierungstechniken zum Einsatz kommen (zusätzlich zu der extremen Form *vertikaler* Partitionierung in Column Stores).

Horizontale Partitionierung (auch „Fragmentierung“ oder „Sharding“) bezeichnet die disjunkte Aufteilung der Zeilen einer (großen) Tabelle auf mehrere (kleine) Tabellen, wobei die Zuweisung einer gegebenen Zeile zu einer der Partitionen über eine system- oder benutzerdefinierte Funktion erfolgt. In einer häufig anzutreffenden Ausprägung von Partitionierung wird jede der Partitionen auf einem anderen Server verwaltet, wodurch einerseits auf jedem einzelnen Server geringere Datenvolumina beherrscht werden müssen und andererseits Geschwindigkeitsvorteile durch Parallelverarbeitung erzielt werden, wie im folgenden Abschnitt thematisiert wird.

In einer zweiten Ausprägung von Partitionierung werden die einzelnen Partitionen auf unterschiedlichen Medien der Speicherhierarchie vorgehalten. Plattner [Pla14a] unterscheidet insbesondere heiße (oder aktive) und kalte (oder passive) Daten, wobei erstere im täglichen Geschäft benötigt werden, während letztere nur noch aus historischen oder regulatorischen Gründen interessant sind. Offenbar ist es unnötig, kalte Daten permanent im Hauptspeicher vorzuhalten; stattdessen können sie auf SSDs oder Festplatten gespeichert werden, wo sie im seltenen Bedarfsfall immer noch verfügbar sind.

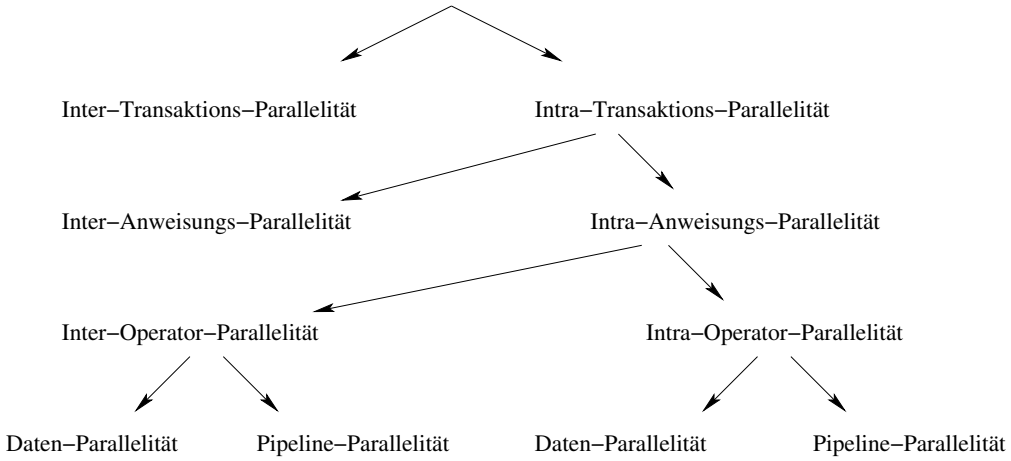

Abbildung 3: Arten von Parallelverarbeitung.

3.4. Parallelverarbeitung und Skalierbarkeit

Wie jedes andere Softwaresystem auch können Hauptspeicherdatenbanken von unterschiedlichen Arten von Parallelismus und unterschiedlicher Hardware-Unterstützung profitieren. Verschiedene Arten von Parallelverarbeitung in Datenbanken werden Rahm [Rah94] folgend in Abbildung 3 zusammengestellt.

In Abbildung 3 werden von oben nach unten die kleiner werdenden Verarbeitungseinheiten einer Datenbank gezeigt, nämlich Transaktionen, die aus mehreren (SQL-) Anweisungen bestehen, welche wiederum durch (Bäume von) Operatoren implementiert werden. Parallelität kann nun sowohl bei der Ausführung einer einzelnen Einheit vorkommen („intra“) als auch bei der Ausführung mehrerer Einheiten („inter“). Zudem liegt Daten-Parallelität vor, wenn Daten partitioniert sind und (Teil-) Operatoren parallel auf unterschiedlichen Partitionen arbeiten. Pipeline-Parallelität bezeichnet demgegenüber eine überlappende Ausführung aufeinanderfolgender (Teil-) Operatoren so, dass nachfolgende Operatoren nicht auf das Ende vorangehender Operatoren warten müssen, sondern ihre Arbeit bereits mit vorliegenden Zwischenergebnissen aufnehmen können.

Die Ausnutzung dieser Arten von Parallelismus erfordert spezielle Implementierungstechniken und kann von unterschiedlicher Hardware-Unterstützung profitieren, nämlich der Parallelverarbeitung (a) innerhalb eines Prozessorkerns, etwa in Form von SIMD-Instruktionen, (b) in verschiedenen Prozessorkernen und (c) in mehreren Servern eines Clusters.

SIMD (Single Instruction, Multiple Data) bezeichnet CPU-Instruktionen, die *Vektoren* von Zahlen anstelle einzelner Zahlen verarbeiten, z. B. die vektorielle Addition mehrerer Zahlen zur beschleunigten Aggregation oder vektorielle logische und Vergleichsoperationen zur beschleunigten Filterung. Spaltenorientiert arbeitende Datenbanken sind in

besonderer Weise für SIMD geeignet, weil die Werte jeder Spalte benachbart im Hauptspeicher abgelegt sind und damit in natürlicher Weise als Vektoren vorliegen. SIMD ermöglicht in obiger Klassifikation also das Ausnutzen von Daten-Parallelität bei Intra-Operator-Parallelität, wobei das Zusammenfassen von Werten zu Vektoren einer (Mikro-) Partitionierung entspricht.

Die Verfügbarkeit mehrerer Prozessorkerne unterstützt im Prinzip jede Parallelitätsart aus Abbildung 3, da jeder Kern mit der Bearbeitung einer anderen Transaktion, einer anderen SQL-Anweisung oder einem anderen (Teil-) Operator betraut sein kann, und zwar entweder unabhängig auf einer eigenen Datenpartition oder in Abhängigkeit der Ergebnisse vorangehender Berechnungen.

Wenn die Leistungsfähigkeit eines gegebenen Systems nicht ausreicht, kommen zwei Arten des Hardware-Ausbaus in Betracht: Zum einen kann die Leistung bestehender Server durch Hinzufügen weiterer Prozessoren oder weiteren Hauptspeichers erhöht werden, was als vertikale Skalierung (engl. *scale up*) bezeichnet wird, keinen größeren Administrationsaufwand mit sich bringt, aber nur begrenzt möglich ist. Zum anderen können im Zuge von horizontaler Skalierung (engl. *scale out*) mehrere Server zu einem Cluster verbunden werden, wobei unterschiedliche Server für unterschiedliche Partitionen der Daten verantwortlich sein können. Hier besteht zwar erhöhter Administrationsaufwand, dafür sind der Leistungsfähigkeit aber kaum noch Grenzen gesetzt.

Verschiedene Datenbanksysteme verfolgen unterschiedliche Strategien bei der Nutzung mehrerer Prozessorkerne und Server. Beispielsweise stellt die mit H-Store [SMA+07] vorangetriebene serielle Transaktionsverarbeitung durch einen einzigen Prozessor-Thread ein Extremum der möglichen Bandbreite dar, während in Hekaton [DFI+13], der Hauptspeicherkomponente von SQL Server, verschiedene Transaktionen auf unterschiedlichen Prozessorkernen ohne Einschränkung parallel ausgeführt werden.

3.5. Weitere Techniken

Aktuelle Hauptspeicherdatenbanken unterscheiden sich in zahlreichen Implementierungstechniken und -details. Besonders augenfällig sind die Unterschiede in der Transaktionsverarbeitung, wobei sich Nebenläufigkeitskontrolle (engl. *concurrency control*) sowie Protokollierung und Wiederherstellung (engl. *recovery*) oftmals drastisch von den bekannten Verfahren traditioneller relationaler Datenbanken unterscheiden.

3.5.1. Nebenläufigkeitskontrolle

Nebenläufigkeitskontrolle verfolgt das Ziel, bei mehreren nebenläufigen oder parallelen Transaktionen eine konsistente Sicht auf den Datenbestand zu gewährleisten und z. B. durch geeignete Synchronisationsverfahren zu verhindern, dass inkonsistente Zwischenergebnisse gelesen oder dass Werte unkontrolliert durch andere Transaktionen überschrieben werden. Das generell akzeptierte Konsistenzkriterium in der Transaktionsverarbeitung wird als *Serialisierbarkeit* bezeichnet (vgl. [WV02, Vos08]) und beruht auf folgender einfachen Idee: Aufgrund des durch die ACID-Garantieen gewährleisteten Konsistenzerhalts einzelner Transaktionen wird die Konsistenz auch bei *serieller* Ausführung von Transaktionen (also jeweils eine neue nach dem Ende einer anderen) gewahrt. Die nebenläufige oder parallele Ausführung einer Menge von Transaktionen wird intuitiv dann als *serialisierbar* bezeichnet, wenn die Interaktionen zwischen den einzelnen Transaktionen auch in einer seriellen (und damit konsistenzerhaltenden) Ausführung hätten vorkommen können.

Um serialisierbare Ausführungen zu gewährleisten, werden Verfahren der Nebenläufigkeitskontrolle eingesetzt. Nebenläufigkeitskontrolle hat allerdings prinzipiell den Nachteil, dass sie den Grad möglicher Parallelität einschränkt und dadurch den erreichbaren Durchsatz (also die pro Zeiteinheit abgearbeitete Anzahl an Transaktionen) reduziert. Entsprechend erlaubt etwa der Sprachstandard SQL auf Konsistenz zu verzichten, indem jede Transaktion in einer von mehreren konfigurierbaren Isolationsstufen (engl. *isolation level*) ausgeführt werden kann. Auf der Stufe *Read Uncommitted* wird auf jegliche Nebenläufigkeitskontrolle verzichtet, was maximale Parallelität erlaubt, allerdings zu korrupten Daten führen kann, während die Stufe *Serializable* höchste Konsistenz in Form von Serialisierbarkeit anstrebt. Unterschiedliche Datenbanksysteme gewährleisten unterschiedliche Isolationsstufen und setzen dazu auf pessimistische oder auf optimistische Verfahren, die nachfolgend kurz skizziert werden. Welche Hauptspeicherdatenbank auf welche Verfahren mit welchen Isolationsstufen setzt, wird in Anhang A zusammengefasst.

Verfahren zur *pessimistischen* Nebenläufigkeitskontrolle vermeiden Inkonsistenzen im Vorfeld, typischerweise durch Sperrverfahren. Eine extreme Form pessimistischer Verfahren besteht schlicht und einfach in der *seriellen* Ausführung der Transaktionen, die mit H-Store [SMA+07] entwickelt und als VoltDB kommerzialisiert wurde und auf der auch HyPer [KN11] basiert. Die serielle Verarbeitung nutzt nur einen einzigen Thread, vermeidet damit jegliche Parallelität, benötigt aber dafür gar keinen Code zur Nebenläufigkeitskontrolle. Um dennoch zumindest einen gewissen Grad an Parallelität zu erzielen, wird die serielle Verarbeitung üblicherweise so mit Partitionierung kombiniert, dass unterschiedliche Threads für separate Partitionen zuständig sind.

Demgegenüber geht *optimistische* Nebenläufigkeitskontrolle davon aus, dass Inkonsistenzen selten genug auftreten, um Transaktionen zunächst mit voller Parallelität arbeiten zu lassen, bevor eine Konsistenzprüfung bei Transaktionsende verifiziert, ob tatsächlich Inkonsistenzen entstanden sind. Derartige Verfahren haben den Nachteil, dass Transaktionen bei negativem Testergebnis abgebrochen werden müssen, nachdem sie ihre eigentliche Arbeit bereits verrichtet haben, was wiederum den erreichbaren Durchsatz reduziert. Ebenso fallen *sperrfreie* (engl. *lock-free* oder *latch-free*) Datenstrukturen wie der Bw-Baum [LLS13], der die Grundlage der Transaktionsverarbeitung in Hekaton bildet, oder die in MemSQL eingesetzten Skip-Listen unter die optimistischen Verfahren.

Mehrversionsnebenläufigkeitskontrolle (MVCC, engl. *multi-version concurrency control*) kann sowohl optimistisch als auch pessimistisch ausgelegt werden, wobei oft Schnappschussisolation (engl. *snapshot isolation*) [BBG+95], welche in [BHF+14] ausführlich für verteilte Szenarien untersucht wird, anstelle der stärkeren Serialisierbarkeit als Konsistenzkriterium gewählt werden kann. Unter MVCC führen Änderungen nicht zum Überschreiben bestehender Daten, sondern erzeugen neue Versionen, die bei erfolgreichem Transaktionsende die bisherigen Versionen ablösen. Schnappschussisolation fordert, dass jeder Transaktion aus logischer Sicht ein Schnappschuss des Datenbestandes zur Verfügung steht, der sämtliche Inhalte bereits abgeschlossener Transaktionen umfasst. Dass Schnappschussisolation ein schwächeres Konsistenzkriterium ist als Serialisierbarkeit, zeigt ein [BBG+95] entlehntes Beispiel: Es gebe zwei miteinander verbundene Konten x und y mit der Integritätsbedingung, dass der summierte Kontostand nicht negativ werden darf, also $x+y \geq 0$. Eingangs gelte $x = y = 50$. Man betrachte die Transaktionen t_x und t_y, die beide Kontostände lesen und dann „ihr“ Konto auf den Wert -40 setzen, wenn die resultierende Summe nicht negativ wird. Wenn beide Transaktionen parallel im Schnappschuss $x = y = 50$ starten, sehen sie Änderungen der jeweils anderen Transaktion nicht und setzen ihr Konto auf -40, wodurch insgesamt der Zustand $x = y = -40$ entsteht, was offenbar die Integritätsbedingung $x+y \geq 0$ verletzt, bei serieller Ausführung aber nicht möglich wäre.

Es sei erwähnt, dass unter Schnappschussisolation neben obiger Anomalie, die auch als *Write Skew* bezeichnet wird, noch weitere Typen von Anomalien auftreten können, wie etwa in [FLO+05] diskutiert wird. Diese Anomalien lassen eine sorgfältige Prüfung angeraten erscheinen, bevor das klassische Konsistenzkriterium der Serialisierbarkeit aus Leistungsgründen abgeschwächt wird. Relativ junge Forschung zeigt, wie Datenbanksysteme, die Schnappschussisolation garantieren, auf das Konsistenzniveau von Serialisierbarkeit angehoben werden können [PG12].

3.5.2. Protokollierung und Wiederherstellung

Recovery bezeichnet die Wiederherstellung eines konsistenten Datenbankzustands nach Fehlerfällen (Soft- und Hardwarefehler sowie Stromausfälle). Diese Wiederherstellung sowie auch die Gewährleistung transaktionaler Dauerhaftigkeit setzen Logging- bzw. Protokollierungsmechanismen während der Transaktionsverarbeitung voraus, damit der Verlust des flüchtigen Hauptspeichers bei Abstürzen oder Stromausfällen durch Log-Daten im stabilen Sekundarspeicher ausgeglichen werden kann.

Recovery-Protokolle unterscheiden sich insbesondere darin, ob sie Undo- oder Redo-Operationen oder beides einsetzen. Undo-Operationen dienen dazu, Effekte abgebrochener Transaktionen rückgängig zu machen, während Redo-Operationen Effekte erfolgreicher Transaktionen (wieder) herstellen. Im Kontext von Hauptspeicherdatenbanken kommen vorwiegend Protokolle mit Redo-Operationen zum Einsatz, um nach einem Fehler sicherzustellen, dass die Effekte aller als erfolgreich gemeldeten Transaktionen sichtbar sind. Demgegenüber sind Undo-Operationen eher selten, weil die Effekte nicht erfolgreicher Transaktionen zuvor nur im Hauptspeicher sichtbar waren und bei einem Neustart von Festplatte daher automatisch nicht mehr existieren.

Eine wichtige Klassifikation von Redo-Protokollen besteht darin, ob im Log geänderte Werte protokolliert werden (wie im Falle der klassischen ARIES-Logging-Familie, vgl. [WV02]) oder die Operationen, die diese Werte produziert haben, was als Command-Logging bezeichnet wird. In [MWMS14] wird am Beispiel der Hauptspeicherdatenbank VoltDB berichtet, dass mit Command-Logging im TPC-C-Benchmark Transaktionsraten erzielt werden können, die diejenigen von ARIES um Faktoren von 1,2 bis 1,5 übertreffen. Dieser Vorteil kommt allerdings zum Preis deutlich erhöhter Recovery-Zeiten (mit Faktoren zwischen 1,5 und 5) im Fehlerfall, da die Transaktionen erneut ausgeführt werden müssen (anstatt einfach die resultierenden Werte zurückzuschreiben.) Command-Logging erscheint demnach in solchen Szenarien attraktiv, in denen hohe Transaktionsraten entscheidend sind, Ausfälle aber selten.

3.5.3. Caching von (Zwischen-) Ergebnissen

Die Wiederverwendung zuvor berechneter Ergebnisse, insbesondere in Form materialisierter Sichten, ist eine seit langem erfolgreich eingesetzte Technik zur Beschleunigung aufwändiger Anfragen. Während die Auswahl und Erzeugung materialisierter Sichten allerdings eine manuelle Optimierungsaufgabe darstellt, die Fachwissen voraussetzt, bringen mittlerweile einige Hauptspeicherdatenbanken verschiedene Techniken zur automatisierten Verwaltung und Nutzung der Ergebnisse vorangegangener Anfragen mit.

Beispielsweise verfügt MonetDB über eine als Recycler [IKNG10] bezeichnete Komponente. Der Recycler verwaltet einen separaten Speicherbereich, in dem der Code und die Ergebnisse einzelner Operatoren, aus denen sich eine komplexe Anfrage zusammensetzt, zwischengespeichert werden. Wenn ein Operator ausgeführt werden soll, dessen Ergebnis sich bereits in diesem Zwischenspeicher befindet (bzw. dessen Ergebnis von einem zwischengespeicherten Ergebnis subsumiert wird), kann die erneute Ausführung des Operators vermieden werden (bzw. deutlich vereinfacht werden).

Neben dieser Caching-Strategie von Zwischenergebnissen können auch die Ergebnisse vollständiger SQL-Anfragen dem Caching unterliegen, wie in [MP13] am Beispiel von SanssouciDB für komplexe OLAP-Anfragen gezeigt wird. SanssouciDB [Pla11] ist eine spaltenorientierte Hauptspeicherdatenbank mit Deltapuffer (vgl. Abschnitt 3.2), welcher für das Ergebnis-Caching besonders geeignet erscheint: Die zwischengespeicherten Ergebnisse beziehen sich auf den Datenbankzustand ohne Berücksichtigung des Deltapuffers. Sie können daher zwar nicht direkt übernommen werden, sondern müssen unter Berücksichtigung des Deltapuffers aktualisiert werden; dafür bleiben die Ergebnisse aber über einen längeren Zeitraum stabil und können daher häufiger wiederverwendet werden.

Beide Systeme zeigen vielversprechende Leistungssteigerungen durch Caching. Eine Untersuchung, welche Caching-Strategie wann empfehlenswert ist, existiert bisher noch nicht.

4. Ausgewählte Hauptspeicherdatenbanksysteme

In den letzten Jahren ist es zu einer Explosion von Datenbanksystemen gekommen, die als Hauptspeicherdatenbanken vermarktet werden. Diese Systeme unterscheiden sich teilweise erheblich, und nicht alle von ihnen sind „echte“ Hauptspeichersysteme im Sinne der in der Einleitung zitierten Definition, die den Hauptspeicher als den primären Speicherort der Daten festlegt. In diesem Abschnitt soll (ohne den Anspruch der Vollständigkeit) die Bandbreite verfügbarer Systeme durch eine Auswahl einflussreicher akademischer und populärer kommerzieller Systeme verdeutlicht werden. Eine längere Auflistung verfügbarer Systeme findet sich beispielsweise in der Wikipedia[11].

Im Folgenden gehen wir zunächst (in näherungsweise chronologischer Reihenfolge) auf die in Forschung und Literatur einflussreichen Datenbanksysteme HyPer, MonetDB und VoltDB ein, bevor wir (in alphabetischer Reihenfolge) Systeme von IBM, MemSQL, Microsoft, Oracle und SAP vorstellen. Eine tabellarische Zusammenfassung der beschriebenen Systeme findet sich in Anhang A.

4.1. MonetDB

MonetDB[12] ist eine seit zwei Jahrzehnten am CWI an der Universität Amsterdam entwickelte Open-Source-Datenbank (lizenziert unter einer Variante der Mozilla Public License, MPL), die großen Einfluss auf die Entwicklung spaltenorientierter Datenhaltung hatte und zahlreiche Innovationen gegenüber traditionellen relationalen Datenbanken mit sich bringt. MonetDB strebt die effiziente Nutzung moderner Hardware einschließlich großer Mengen an Hauptspeicher an, verwaltet die Datenbank aber persistent auf Sekundärspeicher [IGN+12]. Zu den Besonderheiten von MonetDB zählt, dass sie neben relationalen Daten und SQL auch XML und XQuery unterstützt sowie weitere Datenmodelle wie RDF und Arrays integriert werden.

Die auf optimistischer Nebensläufigkeitskontrolle basierende Transaktionsverarbeitung von MonetDB ist hauptsächlich für leseintensive OLAP-Anwendungsszenarien optimiert.[13] Um Transaktionsabbrüche durch nebenläufige Aktualisierungen auf derselben Tabelle zu vermeiden, wird empfohlen, dass Transaktionen intern vom Anwendungsentwickler serialisiert werden. Die Dauerhaftigkeit von Transaktionen wird durch Write-Ahead-Logging (WAL) sichergestellt, wobei die Log-Informationen vor dem Commit auf der Festplatte gespeichert werden. Während des Datenbankstarts sowie in Leerlaufperioden

[11] https://en.wikipedia.org/wiki/List_of_in-memory_databases
[12] https://www.monetdb.org/
[13] https://www.monetdb.org/blog/monetdb-sql-transaction-management-scheme

der Datenbank werden die WAL-Informationen in den primären, persistenten Speicher integriert.

Zu den technischen Neuerungen von MonetDB gehören insbesondere adaptive Optimierungsmechanismen, die den Administrationsaufwand reduzieren sollen. So stellt das in Abschnitt 3.5.3 erläuterte Recycling einen feingranularen Caching-Mechanismus dar, der gemeinsame Teilausdrücke in der Anfrageauswertung erkennt, diese nur einmalig berechnet und im weiteren Verlauf dann auf die zwischengespeicherten Ergebnisse zurückgreift [IKNG10]. Darüber hinaus steht mit Database-Cracking eine adaptive und inkrementelle Indexerstellung zur Verfügung, die mit geringem Mehraufwand den Datenbestand während der Anfrageausführung physisch reorganisiert, um die Ausführung zukünftiger Anfragen zu beschleunigen [IKM07].

Auf physischer Ebene speichert MonetDB Daten persistent spaltenorientiert in binären Assoziationstabellen (BAT), wobei jede BAT die Werte eines Attributs vorhält. Entsprechend wird eine relationale Tabelle mit n Attributen in n BAT aufgeteilt. Intern verwendet MonetDB eine eigene Sprache, die MonetDB Assembly Language (MAL), zur Anfrageauswertung. Die Kernelemente dieser Sprache bilden eine Algebra über BAT und lassen sich durch Array-Operationen effizient implementieren, wobei kompakte Schleifenkonstrukte ein gutes Caching-Verhalten aufweisen sowie Compiler-Optimierungen und spekulative CPU-Ausführungen ermöglichen.

Wie kürzlich in [PPI+14] gezeigt wurde, ist die effiziente Implementierung selbst einfacher Algorithmen unter Berücksichtigung von Caching-, Pipelining- und Spekulationsverzögerungen allerdings nicht-trivial, was großes Optimierungspotenzial für alle Hauptspeicherdatenbanken andeutet. Die Autoren analysieren die in einer naiven Cracking-Implementierung auftretenden Latenzen und erzielen schließlich eine Implementierung, deren Geschwindigkeit (bis zu 25-mal schneller als die naive Implementierung) die eines einfachen Scans erreicht und aufgrund eines besseren Caching-Verhaltens manchmal sogar übertrifft (!).

4.2. VoltDB

VoltDB[14] ist eine Hauptspeicherdatenbank, die aus dem für die serielle Transaktionsverarbeitung wegweisenden (auf GitHub[15] weiterentwickelten) Forschungsprototypen H-Store [SMA+07] hervorgegangen ist. VoltDB ist in einer kommerziellen Enterprise- und einer der Affero General Public License (AGPL) unterliegenden Community-Edition mit

[14] `http://voltdb.com/`
[15] `https://github.com/apavlo/h-store`

eingeschränkter Funktionalität verfügbar. Der Entwicklungsstand von VoltDB im Jahre 2013 wird in [SW13] beschrieben.

VoltDB fokussiert Anwendungsszenarien, in denen Hochverfügbarkeit und ACID-Konsistenz gefordert sind und wo Leseoperationen kleine bis mittlere Datensatzanzahlen umfassen und Schreiboperationen nur wenige Datensätze betreffen, während für komplexe (OLAP-) Anfragen die Einrichtung eines separaten Data-Warehouse-Systems empfohlen wird. Um Hochverfügbarkeit zu garantieren, wird VoltDB in Server-Clustern betrieben, wobei die Daten auf mehrere Server partitioniert werden und jede Partition auf eine konfigurierbare Anzahl anderer Server ($K+1$ Server, um K-Safety zu gewährleisten) repliziert wird. Der Hauptspeicher eines jeden Servers beinhaltet mehrere Partitionen, wobei jede Partition einem anderen CPU-Kern zugewiesen wird.

Transaktionen werden als Stored Procedures in der Datenbank registriert, vorkompiliert und in eine von drei Klassen einsortiert. Erstens sind Single-Node-Transaktionen so gestaltet, dass sie nur eine einzige Partition betreffen. Zweitens umfassen One-Shot-Transaktionen mehrere SQL-Anweisungen, die parallel auf verschiedenen Servern ausgeführt werden können. Die dritte Klasse beinhaltet alle weiteren (allgemeinen) Transaktionen, die also mehrere Server betreffen und Koordination zwischen den Servern erfordern. Ein effizienter Betrieb von VoltDB beruht auf der Annahme, dass Single-Node-Transaktionen den weitaus häufigsten Fall darstellen.

Die Single-Node-Transaktionen werden pro Kern seriell abgearbeitet, ohne dass Sperren oder optimistische Prüfschritte benötigt würden, wodurch theoretisch eine CPU-Auslastung von 100% erzielt werden kann (wenn nur Single-Node-Transaktionen vorkommen und wenn genügend Transaktionen auf unterschiedliche Partitionen entfallen, um alle CPU-Kerne auszulasten). Die anderen Transaktionsarten werden von einer globalen Kontrollinstanz analysiert, aufgeteilt und anteilig in den zugehörigen Partitionsservern im Rahmen der seriellen Verarbeitung ausgeführt.

Replikation kann dank der seriellen Transaktionsverarbeitung einfach dadurch erzielt werden, dass die Transaktionen auf jeder Kopie parallel ausgeführt werden. Durch periodischen Nachrichtenaustausch prüfen die $K+1$ an der Replikation beteiligten Server einer Partition, ob Serverausfälle vorliegen. Nicht antwortende Server werden aus dem Cluster ausgeschlossen, das weiterarbeitet. VoltDB verfügt über einen (deaktivierbaren) Mechanismus, um zu vermeiden, dass verschiedene Server-Gruppen im Falle von Netzwerkpartitionierungen als sich unabhängig entwickelnde Datenbanken weiterarbeiten: Wenn eine Server-Gruppe feststellt, dass sie nicht die Mehrheit[16] der Server umfasst, speichert sie einen Schnappschuss des aktuellen Datenbankinhalts auf Festplatte und fährt

[16] Es wird empfohlen, mit einer ungeraden Anzahl von Servern zu arbeiten.

sich herunter.

VoltDB garantiert die Dauerhaftigkeit der Effekte von Transaktionen auch dann, wenn (z. B. durch einen Stromausfall) das gesamte Server-Cluster ausfällt. Zum einen können (in konfigurierbaren Abständen) transaktionskonsistente Schnappschüsse im Sekundärspeicher gespeichert werden. Zum anderen wird Command-Logging eingesetzt, um erfolgreich abgeschlossene Transaktionen im Sekundärspeicher zu protokollieren. Im Falle eines Ausfalls kann dann zunächst der letzte Schnappschuss wiederhergestellt werden, um anschließend aus dem Command-Log die nach der Schnappschusserstellung abgeschlossenen Transaktionen erneut durchzuführen. Konfigurationsparameter erlauben es, Dauerhaftigkeit gegen Transaktionsdurchsatz abzuwägen (Häufigkeit für Schreiben des Command-Logs in Sekundärspeicher, synchrones oder asynchrones Logging).

4.3. HyPer

HyPer[17] ist eine an der TU München entwickelte Hauptspeicherdatenbank, die OLTP und OLAP unterstützt. Eine Besonderheit von HyPer ist die Nutzung des von der virtuellen Speicherverwaltung des Betriebssystems bereitgestellten Copy-On-Write-Mechanismus zur Isolation von Transaktionen [KN11]. Die eigentlichen Daten werden im Betriebssystem als gemeinsam genutzter Speicher (engl. *shared memory*) verwaltet, der (in erster Näherung) durch seriell ausgeführte OLTP-Transaktionen bearbeitet wird. Parallel dazu werden potenziell lange laufende (OLAP-) Anfragen in eigenen Prozessen ausgeführt. OLAP-Anfragen werden in Kind-Prozessen abgearbeitet, die auf den gemeinsamen Speicher zugreifen. Durch Copy-On-Write arbeitet jede Anfrage automatisch auf einem eigenen Schnappschuss, weil die Speicherverwaltung des Betriebssystems Änderungen durch die OLTP-Transaktionen abfängt und auf einer Kopie des Datenbestandes durchführt. So werden nur lesende Transaktionen automatisch auf transaktionskonsistenten Schnappschüssen ausgeführt, ohne dass Datenbanksynchronisationsmechanismen wie Sperren notwendig wären.

Wie bereits in Abschnitt 3.5 erwähnt können OLTP-Transaktionen parallel ausgeführt werden, wenn sie auf disjunkten Partitionen arbeiten. Die serielle Verarbeitung ist allerdings nicht für länger laufende (oder gar interaktive) Transaktionen geeignet. Diese werden vom System als „bösartig“ erkannt und in einem „vorläufigen“ Modus ausgeführt [MKN13]: Eine derartige Transaktion arbeitet (wie jede OLAP-Anfrage auch) ohne Sperrmechanismen auf ihrem eigenen Schnappschuss. Bei Transaktionsende wird eine von konfigurierbaren Isolationsebenen (View-Serialisierbarkeit und Snapshot-Isolation; von

[17] `http://hyper-db.de/`

(Konflikt-) Serialisierbarkeit ist *keine* Rede) abhängige optimistische Konsistenzprüfung durchgeführt, bevor der modifizierte Schnappschuss in den eigentlichen Datenbestand integriert wird.

Abschließend sei erwähnt, dass SQL-Befehle mit Just-In-Time-Techniken in Maschinensprache übersetzt werden, um Transaktionen möglichst kurz zu halten und dass HyPer Optimierungen für Hardware-Transactional-Memory und NUMA-Architekturen bietet [Neu14]. Bereits in der ursprünglichen HyPer-Publikation aus dem Jahre 2011 [KN11] wurde basierend auf den etablierten OLTP- und OLAP-Benchmarks TPC-C und TPC-H von Transaktionsraten von über 100.000 OLTP-Transaktionen pro Sekunde bei parallelen OLAP-Anfragen auf einem Standard-Server berichtet.

4.4. IBM DB2 BLU

DB2[18] von IBM ist ein seit langem kommerziell sehr erfolgreiches relationales Datenbanksystem. Mit Version 10.5 wurde DB2 unter dem Namen BLU Acceleration um Hauptspeicherfähigkeiten erweitert, was von IBM als zweite Hauptspeicherdatenbank-Generation positioniert wird, nachdem zuvor bereits die Hauptspeicherdatenbank Blink im Informix Warehouse Accelerator zum Einsatz gekommen war [RAB+13]. Laut der Darstellung in [RAB+13] fokussiert BLU primär die Beschleunigung von OLAP-Anfragen durch die in Abschnitt 3 beschriebene spaltenorientierte Datenhaltung gepaart mit Kompression und SIMD-Verarbeitung. Tabellen können wahlweise zeilen- oder spaltenorientiert verwaltet werden, und es ist nicht zwingend erforderlich, dass sämtliche Daten in den Hauptspeicher passen. Die Anfrageverarbeitung ist auf eine effiziente Nutzung von Caches, SIMD-Instruktionen und Multi-Core-Prozessoren ausgelegt.

Mit der Einführung von Fix Pack 10.5.0.4 wurde DB2 zudem um sogenannte Schattentabellen erweitert, wodurch neben OLAP-Anfragen auch die Transaktionsverarbeitung beschleunigt wird, was IBM als OLTAP bezeichnet [Zak14]. Zeilenorientierte Tabellen werden hier durch InfoSphere® Data Replication Change Data Capture kontinuierlich in spaltenoriente (Schatten-) Tabellen repliziert. Änderungen durch Transaktionen wirken zunächst auf zeilenorientierten Tabellen, bevor sie zu den spaltenorientierten Schattentabellen propagiert werden. Im Rahmen der Anfrageverarbeitung entscheidet der DB2-Optimierer je nach Anfrage, ob die Auswertung auf der zeilen- oder der spaltenorientierten Tabelle durchgeführt wird.

DB2 unterstützt durch pessimistische Mehrversionsnebenläufigkeitskontrolle Standard-SQL-Isolationsstufen. Alle Updates in DB2 werden als neue Versionen eingeführt, und

[18] `http://www.ibm.com/software/data/db2/`

Löschungen machen die alten Versionen von Datensätzen ungültig. Durch das geschickte Ausnutzen der Versionen wird die Anzahl benötigter Sperren für Anfragen (und damit der Overhead der pessimistischen Nebensläufigkeitskontrolle) reduziert. Bezüglich der Hochverfügbarkeit bietet IBM DB2 durch Datenredundanz und das Auto-Failover-Verfahren verschiedene Konfigurationen (Idle Standby, mutual takeover, active standbys, und balanced mutual takeover), um höhere Fehlertoleranz für geschäftskritische Anwendungen zu gewährleisten.

4.5. MemSQL

MemSQL[19] ist eine Hauptspeicherdatenbank, die OLTP und OLAP unterstützt. Da uns keine wissenschaftliche Arbeit zu MemSQL bekannt ist, basieren die folgenden Angaben auf Marketing- und Entwicklerinformationen der Web-Seite. Generell ist eine gewisse Vorsicht angeraten, weil viele Behauptungen auf der Web-Seite auf Vergleichen von MemSQL mit MySQL beruhen — also auf Vergleichen mit einer traditionellen zeilenorientierten und festplattenbasierten Datenbank anstelle mit aktuellen Hauptspeicherdatenbanken.

MemSQL stellt als Hauptspeicherdatenbank einen Sonderfall dar, da sie Daten im Hauptspeicher zeilenorientiert verwaltet, während Daten im Flash-Speicher spaltenorientiert abgelegt werden. Daher scheinen keine Kompressionstechniken zur effizienteren Nutzung des Hauptspeichers eingesetzt zu werden.

Durch transaktionskonsistente Schnappschüsse und Transaktionslogs garantiert MemSQL die Dauerhaftigkeit der Daten. Dabei ist konfigurierbar, ob jede Schreibtransaktion vor dem Commit auf der Festplatte aufgezeichnet werden soll (synchrone Dauerhaftigkeit) oder ob nur Hauptspeicher für Dauerhaftigkeit genutzt werden soll, um maximalen Durchsatz beim Schreiben zu erreichen. Zurzeit unterstützt MemSQL ausschließlich die Isolationsstufe *Read Committed*, und jede Anfrage wird individuell in einer separaten Transaktion auf genau einem Kern ausgeführt.[20]

MemSQL setzt auf Mehrversionsnebenläufigkeitskontrolle in Kombination mit sperrfreien Skip-Listen als Standardindex. MemSQL bewirbt insbesondere das auf horizontaler Partitionierung basierende Skalierungsverhalten sowie Hochverfügbarkeit durch Redundanz. SQL-Anweisungen werden zur Leistungssteigerung in Maschinensprache übersetzt, was auf der Web-Seite als Neuerung gegenüber anderen Datenbanken dargestellt wird, aber auch in anderen Systemen (z. B. Hekaton, HyPer und VoltDB) so praktiziert wird.

[19] `http://www.memsql.com/`
[20] `http://developers.memsql.com/docs/latest/faq.html`

4.6. Microsoft SQL Server 2014 Hekaton

Die Hauptspeichertechnologie im SQL Server[21] von Microsoft wird unter dem Code-Namen Hekaton geführt [DFI+13]. Hekaton zielt auf die Beschleunigung von OLTP-Szenarien, indem Tabellen als hauptspeicheroptimierte Hekaton-Tabellen deklariert werden können, die dann dauerhaft im Hauptspeicher vorgehalten werden.

Hekaton verfolgt keine spaltenorientierte Speicherung, sondern implementiert einen optimistischen Mehrversionsnebenläufigkeitskontrolle unter Verzicht auf Sperrverfahren durch zwei Arten sperrfreier, hauptspeicheroptimierter Indexstrukturen in Form von Hash-Tabellen und einer neuen B-Baum-Variante namens Bw-Baum [LLS13]. In einem Demo-Szenario wird in [LLS+14] berichtet, dass der Bw-Baum um den Faktor 3 schneller ist als die in MemSQL verwendete alternative Datenstruktur Skip-Liste.

Hekaton unterstützt die drei Isolationsebenen Schnappschuss, Repeatable Read und Serialisierbarkeit. Zudem werden SQL-Zugriffe auf Hekaton-Tabellen in Maschinensprache übersetzt, um Zeitverluste zu vermeiden. Im Gegensatz zu Systemen mit serieller Verarbeitung wie HyPer und VoltDB setzt Hekaton keine Partitionierung voraus, sondern ermöglicht jedem Thread, ohne Sperrmechanismen auf jede Datenbankzeile zuzugreifen.

Um Dauerhaftigkeit zu erreichen, verwendet Hekaton Transaktionslogs und Checkpoints. Hekaton erzeugt Log-Einträge bei Commit der Transaktionen und fasst mehrere Log-Einträge zu einer größeren Schreib-Operation zusammen. Zudem wird der Checkpointing-Prozess inkrementell und kontinuierlich durchgeführt, um neue Versionen der Daten an eine oder mehrere Datendateien und IDs der gelöschten Versionen an entsprechende Deltadateien anzuhängen. Das Zusammenführen der Datendateien und der entsprechenden Deltadateien wird ausgelöst, wenn der Anteil der nicht gelöschten Versionen in einer Datei unter einen Schwellenwert sinkt. Hekaton unterstützt mit der integrierten AlwaysOn-Komponente nicht nur lokale Hochverfügbarkeit durch Redundanz auf der Server-Instanz-Ebene, sondern auch synchrone oder asynchrone Replikationen zum Zweck von Disaster-Recovery.

4.7. Oracle TimesTen

TimesTen [LNF13] ist eine ursprünglich bei Hewlett-Packard entwickelte und später von Oracle aufgekaufte Hauptspeicherdatenbank, die sowohl als eigenständige Datenbank, als Cache für eine traditionelle Oracle-Datenbank sowie auch als Kern der Exalytics-Appliance von Oracle eingesetzt werden kann. Eine Besonderheit von TimesTen besteht darin, dass die Datenbank neben den üblichen APIs wie ODBC und JDBC ihre Funktionalität auch

[21] `https://www.microsoft.com/en-us/server-cloud/products/sql-server/`

als gemeinsam genutzte Programmbibliothek (engl. *shared library*) zur Verfügung stellt. Daher können entsprechend programmierte Anwendungen die Datenbank über normale Funktionsaufrufe innerhalb ihres Prozesskontextes nutzen, ohne dass Kontextwechsel durch das Betriebssystem nötig wären, was hochperformante Anwendungen ermöglicht.

In TimesTen stehen verschiedene Hash- und Indexstrukturen zur Verfügung, und die Daten können spaltenbasiert unter Dictionary-Encoding komprimiert werden. Per Default laufen Transaktionen auf der Isolationsebene Read Committed, wobei ein Mehrversionsverfahren die Nutzung von Sperren reduziert, aber Inkonsistenzen entstehen können. Die Isolationsebene Serialisierbarkeit steht ebenfalls zur Verfügung, muss aber explizit angefordert werden.

Um Festplattenzugriffe beim Schreiben des Logs zu reduzieren, bietet die Datenbank einen verzögerten Persistenzmodus an, in dem Log-Einträge nicht sofort bei jedem Commit auf die Festplatte geschrieben werden sondern nur alle 100*ms*. Dieser Modus liefert höchste Antwortzeiten, kann bei Systemausfällen aber zum Datenverlust führen.

TimesTen bietet logbasierte Replikationsmechanismen für Hochverfügbarkeit an, wobei Logeinträge erfolgreicher Transaktionen auf einem weiteren Rechner angewendet werden. Unter sogenannter 2-Safe-Replikation werden lokale Transaktionen erst dann erfolgreich abgeschlossen, wenn ihre Ausführung auf einem weiteren Rechner als erfolgreich gemeldet wurde. Diese Replikationsart hat insbesondere das Ziel, die Dauerhaftigkeit von Transaktionen ohne den Einsatz von Festplatten zu erzielen. (Wenn allerdings beide Rechner gleichzeitig ausfallen, können Datenverluste auftreten.) Die in [LNF13] beworbene Konfiguration sieht vor, die gesamte Datenbank in eine Standby-Datenbank zu replizieren, die wiederum die Änderungen an eine Reihe von Read-Only-Datenbanken weiterleitet, auf denen Read-Only-Anwendungen ausgeführt werden.

4.8. Oracle 12c

In der Version 12c hat Oracle[22] seine relationale Datenbank um Hauptspeicherfähigkeiten erweitert. Die Datenbank arbeitet nach wie vor mit Sekundärspeicher als primärem Speicherort, wo Daten zeilenbasiert abgelegt werden. Neu hinzugekommen ist die Möglichkeit, einzelne Tabellen (oder auch nur ausgewählte Attribute oder mehrere Tabellen umfassende Table Spaces) für die hauptspeicherbasierte Verarbeitung zu konfigurieren. Diese Tabellen werden dann aus der zeilenbasierten Repräsentation vom Sekundärspeicher in den Hauptspeicher geladen, wo sie der in Abschnitt 3 beschriebenen spaltenorientierten Datenhaltung gepaart mit Kompression und SIMD-Verarbeitung unterliegen, womit Oracle

[22] `https://www.oracle.com/database/`

analytische Anfragen um Größenordnungen beschleunigt.

Änderungsoperationen werden jedoch nicht an den spaltenorientierten Daten im Hauptspeicher durchgeführt, sondern auf der zeilenorientierten Repräsentation, wobei wie üblich eine Pufferverwaltung für den Transfer der notwendigen Daten zwischen Sekundär- und Hauptspeicher sorgt. Oracle bewirbt dieses Vorgehen als In-Memory-OLTP, wir folgen dieser Charakterisierung jedoch nicht, da der primäre Datenbestand zeilenbasiert im Sekundärspeicher liegt. Durch die zeilenweise Änderung von Daten sollen Leistungssteigerungen für Transaktionen in gemischten Workloads erzielt sowie Datenkonsistenz und Dauerhaftigkeit mit herkömmlichen Transaktionsverarbeitungstechniken gewährleistet werden.

Neben der Beschleunigung von OLAP-Anfragen bewirbt Oracle auch die Beschleunigung von OLTP durch den Verzicht auf Indexstrukturen im Sekundärspeicher: Wenn Indexstrukturen vorhanden sind, müssen sie als redundante Daten im Rahmen der Transaktionsverarbeitung aktualisiert werden, was den Transaktionsdurchsatz reduziert. Da die Hauptspeicherverarbeitung aber schnell genug ist, um ohne Indexstrukturen auszukommen, kann auf diesen Mehraufwand verzichtet werden.

Da die spaltenbasierte In-Memory-Option in die bestehende zeilenbasierte Oracle-Datenbank eingebettet ist, ist sie kompatibel mit vorhandenen Funktionen der Oracle-Datenbank, wodurch Änderungen an den Applikationen nicht erforderlich sein sollen. Durch horizontale Skalierung über mehrere Server lassen sich auch umfangreiche Datenbanken in den Hauptspeicher verlagern. Zudem können selten verwendete Daten auch auf Flash-Speicher oder Festplatten ausgelagert werden.

4.9. SAP HANA DB

HANA DB ist die Hauptspeicherdatenbank der SAP[23]. In die Entwicklung dieser Datenbank sind zahlreiche Forschungsergebnisse eingeflossen, insbesondere unter Leitung von Hasso Plattner am HPI in Potsdam. Viele dieser Ergebnisse sind rund um den Forschungsprototypen SanssouciDB [Pla11] entstanden (und entstehen immer noch). Ein Plädoyer von Plattner aus dem Jahre 2009 zur gemeinsamen Verarbeitung von OLTP und OLAP in einer Hauptspeicherdatenbank unter Vermeidung von Redundanzen und Nutzung der in Abschnitt 3 skizzierten Techniken (spaltenorientierte Datenhaltung mit Deltapuffer, Kompression, Partitionierung) findet sich in [Pla09]; in seinen Büchern [PZ12, Pla14a] werden die Architektur sowie hauptspeicherbasierten Techniken von SanssouciDB detailliert dargestellt.

Die in Abbildung 4 gezeigte Architektur der SAP HANA DB beinhaltet laut [FML+12]

[23] `http://hana.sap.com/`

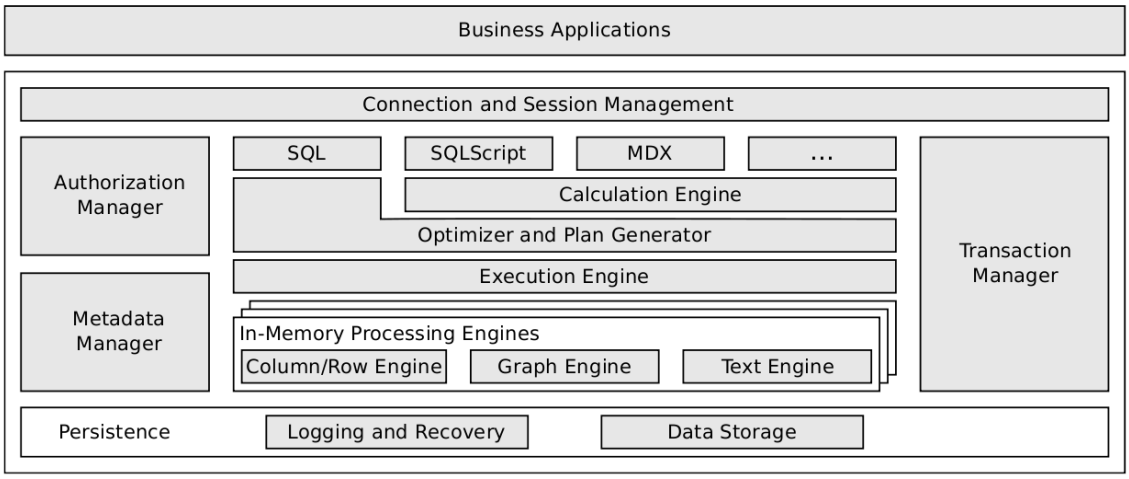

Abbildung 4: Die Architektur der SAP HANA DB (Quelle: [FML+12]).

drei verschiedene Hauptspeicherkomponenten für unterschiedlich strukturierte Daten, nämlich je eine für relationale, für graphbasierte und für textuelle Daten. In der relationalen Komponente existieren wiederum zwei Datenbankkerne, ein spaltenorientierter, der auf der SAP TREX Textsuchmaschine und dem SAP BI Accelerator basiert, und ein zeilenorientierter, der auf P*TIME [CS04] zurückgeht. Die Persistenzfunktionalitäten entstammen schließlich SAP MaxDB. Tabellen können wahlweise zeilen- oder spaltenorientiert angelegt werden.

Diese Komponenten halten die Daten in cache-optimierter und komprimierter Form im Hauptspeicher, falls dieser ausreichend groß ist. Wenn der Hauptspeicher erschöpft ist, werden Tabellen oder Partitionen (durch Anwendungssemantik kontrolliert) ausgelagert.

Die in Abbildung 4 gezeigte Berechnungsmaschine (engl. *calculation engine*) dient der Ausführung von Geschäftslogik innerhalb der Datenbank, ohne Daten in die Anwendungsebene zu transferieren. Sie umfasst eine Bibliothek für Geschäftsfunktionen, um die Geschäftslogik näher an die Daten zu bringen und eine schnellere Entwicklung von Anwendungslogik in der Datenbankebene zu ermöglichen. Außerdem übernimmt sie die Erstellung logischer Berechnungsmodelle, die aus der SAP-eigenen Sprache SQLScript oder anderen domänenspezifischen Sprachen kompiliert werden. Spezielle Operatoren erlauben beispielsweise komplexe statistische Berechnungen und Planungsprozesse innerhalb des Datenbankkerns.

Laut [FML+12] und [LMM+13] implementiert der Transaktionsmanager Snapshot-Isolation oder schwächere Isolationsebenen, wobei Mehrversionsnebenläufigkeitskontrolle in Kombination mit verteilten Sperrverfahren und einem optimierten 2-Phasen-Commit-Protokoll zum Einsatz kommt. Im Hinblick auf die Dauerhaftigkeit der Daten erfasst das Dictionary-Encoding komprimierte Redo-Log alle Änderungen von Datenbanktransaktio-

nen, und wird es vor der Commit auf Festplatte gespeichert. Zusätzliche Mechanismen für Hochverfügbarkeit sowie Disaster-Recovery versprechen eine dauerhaft verfügbare HANA Umgebung für geschäftskritische Anwendungen.

5. Diskussion zu Hauptspeicherdatenbanken für KMU

Der Entscheidung für oder gegen eine Hauptspeicherdatenbank sollte wie bei jeder Software-Einführung eine Anforderungsanalyse vorangehen. Zwei mögliche Ausgangspunkte liegen dabei in der Unterscheidung von (1) Migration und (2) Neuentwicklung.

Migration Eine Migration beruht auf einer existierender Datenbankanwendung, deren Leistung für eine angemessene Unterstützung des Anwendungsziels (sei es im Tagesgeschäft oder im Rahmen von Entscheidungsunterstützung) zu niedrig ist und wo der Wechsel zu Hauptspeichertechnologie einen Leistungssprung verspricht. In dieser Situation müssen zunächst die Gründe für die bisherigen Leistungsprobleme identifiziert werden, die auch außerhalb der Datenbank liegen können (im Kontext einer typischen Mehrschichten-Client-Server-Architektur können in jeder Schicht und zwischen allen benachbarten Schichten Engpässe auftreten). Wenn die Datenbank selbst denn Engpass darstellt, kommen neben dem Umstieg auf eine Hauptspeicherdatenbank natürlich klassische Tuning- und Skalierungs-Techniken in Betracht, die im Rahmen einer Kosten-Nutzen-Analyse verglichen werden sollten. Der Einsatz klassischer Tuning-Techniken (wie Indexerstellung, Materialisierung, Denormalisierung, Fragmentierung, vgl. [SB02]) erfordert in erster Linie Expertise, die im Unternehmen vorhanden sein kann oder auch nicht, während das Aufrüsten der bestehenden Hardware (scale up oder scale out, je nach Fähigkeiten des bestehenden Datenbanksystems) eine einmalige Investition bedeutet.

Falls die Wahl auf eine Hauptspeicherdatenbank fällt, sind mehrere Szenarien denkbar. Im einfachsten Fall löst die Hauptspeicherdatenbank mit ihrem in Abschnitt 2.1 besprochenen Geschwindigkeitsvorteil eine bestehende, festplattenbasierte Datenbank ab. Komplizierter wird es, wenn Berechnungen aus der Anwendungsschicht zur Datenbank verschoben werden müssen, um eine zufriedenstellende Leistung zu erzielen. Je nach aktueller Anwendungsarchitektur kann dies durch einfache Änderungen am Code zu realisieren sein oder auch ein vollständiges Re-Design erfordern. Wir kommen weiter unten auf dieses Thema zurück.

Neuentwicklung Typische Szenarien für Neuentwicklungen mit Hauptspeicherdatenbanken finden sich wie in Abschnitt 2.2 skizziert im Big-Data-Umfeld: Zum einen sind dies Anwendungen, die auf umfangreichen Datenbeständen aufbauen und die kurze Antwortzeiten oder interaktive Analysen erfordern, die also die Charakteristika Volumen und Geschwindigkeit von Big Data aufweisen. Zum anderen ermöglichen erweiterte Features

für datenbankinterne Geschäftslogik, für analytische Berechnungen und Modelle sowie für semi- oder unstrukturierte Daten Anwendungen mit den Big-Data-Charakteristika Vielfalt und Wert. Je nach Anwendungsfall ist dann eine sorgfältige Auswahl eines Zielsystems mit den erforderlichen Fähigkeiten notwendig.

Herstellerauswahl Die Auswahl eines Herstellers und Zielsystems sollte sowohl technische als auch nicht-technische Faktoren einbeziehen. Aus technischer Perspektive ist zu prüfen, welche Systeme Unterstützung für spezielle Anforderungen mitbringen. Das Fehlen einer speziellen Fähigkeit in einem Hauptspeicherdatenbanksystem muss noch nicht sofort dessen Ausschluss bedeuten, weist aber darauf hin, dass dann bei Implementierung oder Migration ein erhöhter Entwicklungsaufwand zu erwarten ist. Beispielsweise können weit über den SQL-Standard hinausgehende Funktionalitäten wie Bibliotheken mit Funktionen für analytische Vorhersagemodelle den Implementierungsaufwand deutlich reduzieren, allerdings nur wenn sie für das gegebene Szenario relevant sind. Zudem sollten Anforderungen an die Interoperabilität mit existierenden Systemen spezifiziert und überprüft werden (z. B. die Nutzung herstellerspezifischer Sprachdialekte in Anwendungen oder die Verwendung von Schnittstellen zu Anwendungssystemen wie CRM- oder ERP-Systemen). Hier ist zu erwarten, dass die Nutzung einer Hauptspeicherdatenbank mit einem Anwendungssystem aus der Hand eines einzelnen Herstellers unproblematischer verläuft als bei Best-of-Breed-Kombinationen. Darüber hinaus ist zu bedenken, dass nicht alle Lösungen gleichermaßen für KMU wie für Großunternehmen geeignet sind: Wie Grabova et al. [GDCZ10] für den Kontext von Business-Intelligence-Projekten festhalten, bieten sich für KMU leichtgewichtige, kostengünstige und flexible Web basierte Lösungen an.

Zwei zentrale Aspekte zur Auswahl eines Herstellers aus nicht-technischer Perspektive sind einerseits die Kosten für die Einführung der Hauptspeicherdatenbank (Erwerb oder Miete von Hardware, möglicherweise im Rahmen einer Cloud-Lösung, Entwicklungs- und ggf. Lizenzkosten) und andererseits die Reputation des bzw. strategische Beziehung zum Hersteller.

Cloud-Lösungen Obwohl ein Großteil der Angebote von Hauptspeicherdatenbanken unterstellt, dass eigene, den Spezifikationen des Herstellers entsprechende Hardware zum Einsatz kommt, sind auch Software-as-a-Service-Angebote (SaaS) „in der Cloud“ verfügbar, und zwar in den üblichen Varianten von öffentlicher, privater oder hybrider Cloud, teilweise auch in herstellereigenen Rechenzentren. Fragen rund um Themen der Cloud-Nutzung durch KMU werden seit Längerem diskutiert. Wir verweisen hier auf

[HV10] als Handreichung für den Datenbankbetrieb in der Cloud sowie auf [HV11] als frühe Bestandsaufnahme der SaaS-Nutzung durch KMU. Verkürzt dargestellt verspricht der Einsatz einer Cloud-Lösung niedrige Einstiegsbarrieren hinsichtlich Kosten (da ohne eigene Hardware nutzungsbasiert abgerechnet wird) und Expertise (da Installation, Betrieb und Wartung in externer Verantwortung liegen), setzt aber ein gewisses Vertrauen in den Betreiber der Cloud-Infrastruktur voraus, was gerade von KMU kritisch beurteilt wird. Um dieser Problematik zu begegnen, wird in [Has12] eine Cloud-Strategie für KMU mit einem Ansatz zur Vertrauensbildung durch genossenschaftliche Verbünde von KMU entwickelt.

Vereinfachung der IT-Architektur Unabhängig von der Frage, ob die notwendige Hardware aus eigenen Servern besteht oder in der Cloud genutzt wird, kann die Einführung einer Hauptspeicherdatenbank den Anlass bieten, über mögliche Vereinfachungen der IT-Architektur sowie Veränderungen im Anwendungsentwurf nachzudenken. Im günstigsten, in Abschnitt 2.1 beschriebenen Fall kann die bestehende Datenbank mit deutlichen Geschwindigkeitsvorteilen durch eine Hauptspeicherdatenbank abgelöst werden, ohne dass Änderungen an Anwendungen erforderlich sind. Zudem kann die in Abschnitt 2.3 dargestellte Vereinfachung der IT-Architektur durch Zusammenlegung von OLTP und OLAP auf einem leistungsfähigen Hauptspeicherdatenbanksystem helfen, Komplexität und Kosten zu reduzieren.

Es ist allerdings zu bedenken, dass die größtmögliche Leistung mit Hauptspeicherdatenbanken nur dann erzielt werden kann, wenn folgende zwei Prinzipien berücksichtigt werden. Erstens ist es leistungsfördernd, den Code zu den Daten zu bringen: Es ist deutlich effizienter, Berechnungen (auch Geschäftslogik) innerhalb des Datenbankkerns selbst auszuführen und lediglich kleine Resultate zur Anwendungsschicht zu übertragen, als große Datenmengen zwischen Datenbankserver und Anwendungsserver zu transferieren. Zweitens sollte das Datenbankschema möglichst redundanzfrei ausgelegt werden, um die Anzahl notwendiger Updates zu reduzieren: Zum einen reicht (wie Plattner in [Pla09] argumentiert und wie wir am Beispiel von Oracle in Abschnitt 4.8 erläutert haben) die Leistung von Hauptspeicherdatenbanken oftmals aus, um auf typische Tuning-Techniken wie Indexstrukturen und materialisierte Sichten verzichten zu können, was den Mehraufwand zur Wartung dieser redundanten Datenstrukturen im Rahmen der Transaktionsverarbeitung eliminiert. Zum anderen plädiert Plattner in [Pla14b] dafür, in spaltenorientiert ausgelegten Systemen im Datenbankschema generell auf Aggregatwerte zu verzichten. Ein typisches Beispiel solcher Aggregatwerte sind Kontostände von Kunden. Wenn der Kontostand als separates Attribut gespeichert wird, erfordert dies bei jeder Buchung zusätzlich zu der Einfüge-Operation für die Buchung selbst eine Update-Operation zur Aktualisie-

rung des aggregierten Kontostandes, die insbesondere in spaltenorientierten Systemen zu erhöhtem Mehraufwand führt. Wenn der Kontostand demgegenüber nicht mehr gespeichert, sondern nur noch bei Bedarf aus den Einzelbuchungen aggregiert wird, kann auf Update-Operationen verzichtet werden, was die Transaktionsverarbeitung vereinfacht und beschleunigt.

Entsprechend hängt das Potential der Leistungsverbesserung durch die Einführung von Hauptspeicherdatenbanken stark vom Zuschnitt der Anwendungen und der verfügbaren Entwicklungskapazität ab. Wenn größere Anpassungen an den Anwendungen erforderlich sind, kann gerade in KMU der dann erforderliche Entwicklungsaufwand die internen Ressourcen übersteigen. In dieser Situation kann entweder nur noch die mit Hauptspeicherdatenbanken einhergehende grundlegende Leistungsverbesserung erzielt werden, oder eine (Neu-) Entwicklung muss mit externen Partnern durchgeführt werden.

Einstiegsbarrieren für KMU Schließlich ist zu bedenken, dass Einstiegsbarrieren für KMU bei der Einführung von Hauptspeicherdatenbanken denjenigen von Business-Intelligence-Anwendungen ähnlich sein dürften (vgl. [GDCZ10]): Zunächst ist eine Kombination von kaufmännischer und technische Expertise erforderlich, um ein überzeugendes Geschäftsszenario auf Basis von Hauptspeicherdatenbanken zu identifizieren. Die in Abschnitt 2 vorgestellten Beispiele mögen hier erste Ideen liefern. Aus organisatorischer Perspektive ist zudem die Unterstützung durch das Management erforderlich, um ein Migrations- oder Einführungsprojekt mit ausreichendem Budget für Soft- und Hardware, Implementierung und Aufbau notwendiger Expertise auf den Weg zu bringen. Aus technischer Perspektive ist insbesondere die Integration einer Hauptspeicherdatenbank in bestehende Anwendungen eine komplexe Herausforderung für KMU, weil hier im Regelfall die Zusammenarbeit mit heterogenen Anbietern auf den einzelnen Ebenen der Anwendungsarchitektur unabdingbar ist.

6. Zusammenfassung und Ausblick

Hauptspeicherdatenbanken haben in den vergangenen Jahren dramatisch an Popularität gewonnen, und zahlreiche Hersteller bieten mittlerweile Systeme an, die durch spezielle Anfrage- und Transaktionsverarbeitungstechniken die effiziente Datenverarbeitung im Hauptspeicher anstreben und damit die Festplatte als primären Speicherort der Daten durch den Hauptspeicher ersetzen, was Latenzen eliminiert und einen Leistungssprung in der Datenverarbeitung verspricht. In diesem Bericht haben wir Anwendungsbeispiele für derartige Systeme skizziert, ihre technischen Grundlagen erläutert, exemplarische Systeme mit ihren Besonderheiten vorgestellt sowie Herausforderungen für den Einsatz von Hauptspeicherdatenbanken diskutiert.

Bezüglich des Nutzens von Hauptspeicherdatenbanken für KMU zeigt sich noch kein klares Bild. Echte Erfolgsgeschichten fehlen hier noch, und auch unter den in Abschnitt 2 skizzierten Anwendungsbeispielen findet sich kein „echtes“ KMU. Dennoch lässt sich festhalten, dass mittlerweile ein genereller Trend zu Hauptspeicherdatenbanken besteht, nachdem die hauptspeicherbasierte Datenverarbeitung zunächst lange Zeit eine Domäne spezialisierter Datenbanksysteme war: Die drei großen Hersteller traditioneller relationaler Datenbanken, IBM, Microsoft und Oracle, bieten für ihre Flagschiffe mittlerweile den Betrieb als Hauptspeicherdatenbank zumindest als Option an. Dementsprechend wird es immer einfacher, das Leistungsverhalten einer Anwendung bei Umstieg auf eine Hauptspeicherdatenbank zumindest prototypisch zu testen. Für kostengünstige oder prototypische Neuentwicklungen kann zudem auf freie Software (MonetDB, Community Edition von VoltDB) zurückgegriffen werden.

Literatur

[BBG+95] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.*, pages 1–10. ACM Press, 1995.

[BHF+14] Carsten Binnig, Stefan Hildenbrand, Franz Färber, Donald Kossmann, Juchang Lee, and Norman May. Distributed snapshot isolation: global transactions pay globally, local transactions pay locally. *VLDB J.*, 23(6):987–1011, 2014.

[CS04] Sang Kyun Cha and Changbin Song. P*time: Highly scalable OLTP DBMS for managing update-intensive stream workload. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 1033–1044. Morgan Kaufmann, 2004.

[DFI+13] Cristian Diaconu, Craig Freedman, Erik Ismert, Per-Åke Larson, Pravin Mittal, Ryan Stonecipher, Nitin Verma, and Mike Zwilling. Hekaton: SQL server's memory-optimized OLTP engine. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1243–1254. ACM, 2013.

[DKO+84] David J. DeWitt, Randy H. Katz, Frank Olken, Leonard D. Shapiro, Michael Stonebraker, and David A. Wood. Implementation techniques for main memory database systems. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 1–8. ACM Press, 1984.

[Eic89] Margaret H. Eich. Main memory database research directions. In Haran Boral and Pascal Faudemay, editors, *Database Machines, Sixth International Workshop, IWDM '89, Deauville, France, June 19-21, 1989, Proceedings*, volume 368 of *Lecture Notes in Computer Science*, pages 251–268. Springer, 1989.

[FLO+05] Alan Fekete, Dimitrios Liarokapis, Elizabeth J. O'Neil, Patrick E. O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.

[FML+12] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[GDCZ10] Oksana Grabova, Jerome Darmont, Jean-Hugues Chauchat, and Iryna Zolotaryova. Business intelligence for small and middle-sized entreprises. *SIGMOD Rec.*, 39(2):39–50, December 2010.

[GLV84] Hector Garcia-Molina, Richard J. Lipton, and Jacobo Valdes. A massive memory machine. *IEEE Trans. Computers*, 33(5):391–399, 1984.

[HAMS08] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. OLTP through the looking glass, and what we found there. In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 981–992. ACM, 2008.

[Has12] Till Haselmann. *Cloud-Services in kleinen und mittleren Unternehmen: Nutzen, Vorgehen, Kosten*. PhD thesis, Westfälische Wilhelms-Universtität Münster, 2012. Wissenschaftliche Schriften der WWU Münster, Reihe IV, Band 6.

[HV10] Till Haselmann and Gottfried Vossen. Database-as-a-Service für kleine und mittlere Unternehmen. Technical report, Arbeitsberichte des Instituts für angewandte Informatik (IAI), Münster, 2010.

[HV11] Till Haselmann and Gottfried Vossen. Software-as-a-service in small and medium enterprises: An empirical attitude assessment. In Athman Bouguettaya, Manfred Hauswirth, and Ling Liu, editors, *Web Information System Engineering WISE 2011*, volume 6997 of *Lecture Notes in Computer Science*, pages 43–56. Springer Berlin Heidelberg, 2011.

[IGN+12] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, and Martin L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[IKM07] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 68–78. www.cidrdb.org, 2007.

[IKNG10] Milena G. Ivanova, Martin L. Kersten, Niels J. Nes, and Romulo A.P. Gonçalves. An architecture for recycling intermediates in a column-store. *ACM Trans. Database Syst.*, 35(4):24:1–24:43, October 2010.

[KN11] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 195–206. IEEE Computer Society, 2011.

[LLS13] Justin J. Levandoski, David B. Lomet, and Sudipta Sengupta. The Bw-Tree: A B-tree for new hardware platforms. In Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 302–313. IEEE Computer Society, 2013.

[LLS+14] Justin J. Levandoski, David B. Lomet, Sudipta Sengupta, Adrian Birka, and Cristian Diaconu. Indexing on modern hardware: hekaton and beyond. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 717–720. ACM, 2014.

[LMM+13] Juchang Lee, Michael Muehle, Norman May, Franz Faerber, Vishal Sikka, Hasso Plattner, Jens Krüger, and Martin Grund. High-performance transaction processing in SAP HANA. *IEEE Data Eng. Bull.*, 36(2):28–33, 2013.

[LNF13] Tirthankar Lahiri, Marie-Anne Neimat, and Steve Folkman. Oracle TimesTen: An in-memory database for enterprise applications. *IEEE Data Eng. Bull.*, 36(2):6–13, 2013.

[MKN13] Henrik Mühe, Alfons Kemper, and Thomas Neumann. Executing long-running transactions in synchronization-free main memory database sys-

tems. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013.

[MP13] Stephan Müller and Hasso Plattner. Aggregates caching in columnar in-memory databases. In Arun Jagatheesan, Justin J. Levandoski, and Thomas Neumann, editors, *Proceedings of the 1st International Workshop on In Memory Data Management and Analytics, IMDM 2013, Riva Del Garda, Italy, August 26, 2013.*, pages 62–73, 2013.

[MWMS14] Nirmesh Malviya, Ariel Weisberg, Samuel Madden, and Michael Stonebraker. Rethinking main memory OLTP recovery. In Isabel F. Cruz, Elena Ferrari, Yufei Tao, Elisa Bertino, and Goce Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 604–615. IEEE, 2014.

[Neu14] Thomas Neumann. Engineering high-performance database engines. *PVLDB*, 7(13):1734–1741, 2014.

[PG12] Dan R. K. Ports and Kevin Grittner. Serializable snapshot isolation in postgresql. *PVLDB*, 5(12):1850–1861, 2012.

[Pla09] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 1–2. ACM, 2009.

[Pla11] Hasso Plattner. Sanssoucidb: An in-memory database for processing enterprise workloads. In Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, and Holger Schwarz, editors, *Datenbanksysteme für Business, Technologie und Web (BTW), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme"(DBIS), 2.-4.3.2011 in Kaiserslautern, Germany*, volume 180 of *LNI*, pages 2–21. GI, 2011.

[Pla14a] Hasso Plattner. *A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases*. 2 edition, 2013, 2014.

[Pla14b] Hasso Plattner. The impact of columnar in-memory databases on enterprise systems. *PVLDB*, 7(13):1722–1729, 2014.

[PPI+14] Holger Pirk, Eleni Petraki, Stratos Idreos, Stefan Manegold, and Martin L. Kersten. Database cracking: fancy scan, not poor man's sort! In Alfons Kemper and Ippokratis Pandis, editors, *Tenth International Workshop on Data Management on New Hardware, DaMoN 2014, Snowbird, UT, USA, June 23, 2014*, page 4. ACM, 2014.

[PZ12] Hasso Plattner and Alexander Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2 edition, 2012.

[RAB+13] Vijayshankar Raman, Gopi K. Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, Shaorong Liu, Guy M. Lohman, Tim Malkemus, René Müller, Ippokratis Pandis, Berni Schiefer, David Sharpe, Richard Sidle, Adam J. Storm, and Liping Zhang. DB2 with BLU acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.

[Rah94] Erhard Rahm. *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung*. Addison-Wesley, 1994.

[SB02] Dennis E. Shasha and Philippe Bonnet. *Database Tuning - Principles, Experiments, and Troubleshooting Techniques*. Elsevier, 2002.

[SKZP10] Matthieu-P. Schapranow, Ralph Kühne, Alexander Zeier, and Hasso Plattner. Enabling real-time charging for smart grid infrastructures using in-memory databases. In *The 35th Annual IEEE Conference on Local Computer Networks, LCN 2010, 10-14 October 2010, Denver, Colorado, USA, Proceedings*, pages 1040–1045. IEEE, 2010.

[SMA+07] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The end of an architectural era (it's time for a complete rewrite). In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1150–1160. ACM, 2007.

[SW13] Michael Stonebraker and Ariel Weisberg. The VoltDB main memory DBMS. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[Tas14] Rene Tassy. Improving transactional scalability in MonetDB. Master's thesis, University of Amsterdam, 2014.

[Vos08] Gottfried Vossen. *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 5. Auflage*. Oldenbourg, 2008.

[Vos14] Gottfried Vossen. Big Data: Der neue Katalysator für Business und andere Intelligenz. In Torsten Schwarz, editor, *Leitfaden Marketing Automation*, pages 1–12. 2014. Im Druck.

[WV02] Gerhard Weikum and Gottfried Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.

[Zak14] Ayesha Zaka. Improve performance of mixed OLTAP workloads with DB2 shadow tables. Technical report, IBM Corporation, 2014. developerWorks.

A. Tabellarische Übersichten

Zentrale Eigenschaften der in Abschnitt 4 diskutierten Hauptspeicherdatenbanksysteme werden in Tabelle 1 zusammengefasst. Dort steht das Symbol „✔“ für explizite Unterstützung einer Eigenschaft und „✖“ für eine fehlende Eigenschaft. Das Symbol „**–**“ tritt lediglich bezogen auf hauptspeicherbasiertes OLTP und OLAP auf und kennzeichnet dort Systeme, die nicht explizit für das jeweilige Szenario ausgelegt und optimiert sind, es aber nicht ausschließen.

In Tabelle 2 werden zentrale, von den verschiedenen Systemen unterstützte Isolationsstufen zusammengefasst. Für die generelle Bedeutung von Isolationsstufen als Kompromiss zwischen Konsistenz und Leistung sei auf Abschnitt 3.5.1 verwiesen. Folgende Aspekte seien betont bzw. näher erläutert: Für MonetDB gibt lediglich die Masterarbeit [Tas14] an, dass sämtliche Isolationsstufen des SQL-Standards unterstützt würden, während offizielle Dokumentation und wissenschaftliche Publikationen nur die Isolationsstufen unterhalb von Serialisierbarkeit stützen, weshalb in der Tabelle das Symbol „**?**“ für Serialisierbarkeit eingetragen ist. Bezüglich HyPer sei aus Abschnitt 4.3 daran erinnert, dass verschiedene Transaktionstypen unterschieden werden, die unterschiedlich behandelt werden, mit View-Serialisierbarkeit für „bösartige“ Transaktionen. VoltDB geht als einziges System keinerlei Kompromisse bezüglich der Konsistenz ein und garantiert in jedem Fall Serialisierbarkeit. DB2 BLU erzielt Serialisierbarkeit über den in Abschnitt 4.4 erläuterten „Umweg“ der zeilenbasierten Repräsentation.

Insgesamt ergibt sich aus Tabelle 2 ein uneinheitliches Bild, wobei mit der Voreinstellung *Read Committed* in vielen Systemen ein relativ niedriges Konsistenzniveau vorgeschlagen wird. Das Studium der jeweils aktuellsten Version der Entwicklerdokumentation erscheint angeraten.

Tabelle 1: Übersicht der diskutierten Hauptspeicherdatenbanksysteme.

	Lizenz	OLTP In-Memory	OLAP In-Memory	Hauptspeicher spaltenorientiert	Nebenläufigkeits-kontrolle	Skalierbar, hochverfüg-bar	Cloud
MonetDB	Frei (MPL-Variante)	–	✔	✔	Optimistisch	✖	✖
VoltDB	Frei (AGPL), kommerziell	✔	–	✖	Serielle Ausführung	✔	✔
HyPer	Akademisch	✔	✔	✔	Copy-On-Write (OLAP); serielle Ausführung (OLTP)	✖	✖
IBM DB2 BLU	Kommerziell	✔	✔	✔	Sperren	✔	✔
MemSQL	Kommerziell	✔	✔	✖	Sperrfreie Indexstrukturen	✔	✔
MS SQL Server 2014 Hekaton	Kommerziell	✔	–	✖	Sperrfreie Indexstrukturen und optimistisch	✔	✔
Oracle TimesTen	Kommerziell	✔	✔	✔	Sperren	✔	✔
Oracle 12c	Kommerziell	✔	✔	✔	Sperren	✔	✔
SAP HANA	Kommerziell	✔	✔	✔	Sperren	✔	✔

Legende: ✔ – Unterstützt; – – Ohne spezielle Unterstützung; ✖ – Nicht vorhanden

Tabelle 2: Isolationsstufen der diskutierten Hauptspeicherdatenbanksysteme.

	Read Uncommitted	Read Committed	Schnappschuss-Isolation	Serialisierbarkeit
MonetDB	✔	✔	✔	?
VoltDB	✖	✖	✖	✔
HyPer	✖	✖	✔	✔
IBM DB2 BLU	✔	✔(default)	✖	zeilenbasiert
MemSQL	✖	✔	✖	✖
MS SQL Server 2014 Hekaton	✖	✔(default)	✔	✔
Oracle TimesTen	✖	✔(default)	✖	✔
Oracle 12c	✖	✔(default)	✔	✖
SAP HANA	✖	✔(default)	✔	✖