KitMig – Flexible Live-Migration in mandantenfähigen Datenbanksystemen

Andreas Göbel

Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2
07743 Jena
andreas.goebel@uni-jena.de

Marcel Sufryd

Wirtschaftswissenschaftliche Fakultät
Friedrich-Schiller-Universität Jena
Carl-Zeiss-Straße 3
07743 Jena
marcel@sufryd.com

Abstract: Mandantenfähige Datenbanksysteme ermöglichen die gemeinsame Nutzung physischer Ressourcen durch eine Vielzahl von Mandanten. Ihr Einsatz erlaubt Anbietern von Cloud-Datenbankdiensten die Reduzierung der Betriebskosten durch eine hohe Ressourcennutzung und die Ausnutzung von Skaleneffekten. Die Migration von Mandanten innerhalb einer Serverfarm erweist sich in mandantenfähigen Datenbanksystemen als eine Schlüsselkomponente für Elastizität, Lastverteilung und Wartbarkeit. In diesen Einsatzbereichen werden jedoch unterschiedliche und zum Teil unvereinbare Anforderungen an eine Migration gestellt. Existierende Ansätze zur Live-Migration eignen sich aufgrund ihres statischen Ablaufs nur in wenigen Fällen.

In diesem Beitrag stellen wir das Framework KitMig zur Live-Migration in mandantenfähigen Datenbanksystemen vor. In Anlehnung an einen Baukasten stellt es verschiedene Module zur Bestimmung des Migrationsablaufs bereit. Die geeignete Kombination von Modulen erlaubt die Anpassung des Ablaufs an die gestellten Anforderungen. Im Rahmen des Beitrags werden die KitMig-Phasen, zugehörige Module und die Implementierung im Open-Source-DBMS H2[1] beschrieben. Mehrere Untersuchungen demonstrieren die Charakteristik verschiedener Modulkombinationen und die Anwendbarkeit der resultierenden Abläufe in ausgewählten Einsatzbereichen.

1 Einleitung

Relationale Datenbanksysteme sind zentraler Bestandteil des Software-Stacks vieler Unternehmen. Die Anbieter von Cloud-Datenbankdiensten (Database as a Service, DbaaS [SK12]) verfolgen das Ziel, Unternehmen eine Alternative zum Betrieb der Datenbanksysteme innerhalb des Unternehmens zu bieten. Angebote wie Windows Azure SQL-Datenbank[2] oder Amazon Relational Database Service (RDS)[3] sorgen beispielsweise für die Bereitstellung, Provisionierung, Konfiguration, Überwachung und Wartung von Datenbanken. Zu den Herausforderungen dieser Dienste gehört die adäquate Verwaltung einer großen Anzahl von Mandanten (Unternehmen, Organisationen oder Endkunden)

[1] http://www.h2database.com/
[2] http://www.windowsazure.com/services/sql-database/
[3] http://aws.amazon.com/rds/

mit vielfältigen und kaum vorhersehbaren Zugriffsmustern. Durch die Konsolidierung von Mandanten mit entgegengesetzten Aktivitätszeiträumen und Ressourcenanforderungen vermeiden Dienstanbieter die stetige Bereitstellung von Ressourcen für die maximale Last jedes einzelnen Mandanten [JA07]. Die Nutzung von Skaleneffekten als Folge höherer Ressourcenausnutzung und vereinfachter Wartung erlaubt ihnen die Reduzierung der Betriebskosten beim Einsatz dieser mandantenfähigen Datenbanksysteme. Die Anzahl der zu betreibenden und administrierenden Datenbanksysteminstanzen können durch sie deutlich reduziert werden. Ihre Einsatzbereiche sind nicht auf DbaaS beschränkt, sie sind beispielsweise ein entscheidendes Mittel bei der Datenverwaltung in den Bereichen Software-as-a-Service (SaaS) und Platform-as-a-Service (PaaS).

Die Implementierungsvarianten mandantenfähiger Datenbanksysteme verdeutlichen den Konflikt zwischen einem hohen Isolationsgrad der Mandantendaten und hoher Ressourcenausnutzung. Jacobs und Aulbach [JA07] unterscheiden drei wesentliche Konsolidierungsansätze:

- *Shared Machine:* Die Mandanten verwenden ein gemeinsames Datenbanksystem, besitzen jedoch separate Datenbanken.
- *Shared Process:* Die Mandanten teilen sich Datenbankprozesse, besitzen jedoch getrennte Tabellen.
- *Shared Table:* Die Daten mehrerer Mandanten liegen in gemeinsamen Tabellen.

Einige kommerzielle Systeme wie Amazon RDS sowie Forschungsprojekte wie Relational Cloud [CJP+11] setzen aufgrund der hohen Mandantenisolation auf den Ansatz *Shared Machine*. Die Isolation verursacht jedoch hohe Kosten aufgrund geringer Ressourcenauslastung und nicht realisierbarer mandantenübergreifender Administration. Dieser Effekt wird bei anknüpfenden Ansätzen wie separaten Datenbankinstanzen oder separaten virtuellen Maschinen (VMs) abermals verstärkt [Rei10]. Der Ansatz *Shared Table* erlaubt hingegen eine sehr effiziente Ressourcennutzung. Dienstanbieter müssen durch eine geeignete Abbildung die vielfältigen Mandantendaten in einem physischen Datenbankschema vereinen und zugleich die Isolation der Daten und der Performance von Mandanten gewährleisten [AJKS09]. Individualität bei der Administration und DBMS-Funktionalität kann hierbei in der Regel nicht geboten werden. Der Ansatz ist daher insbesondere bei Mandanten mit vergleichbaren Tabellenstrukturen, Zugriffsmustern und Anforderungsprofilen geeignet, was ihn beispielsweise für SaaS-Anbieter wie Salesforce.com [WB09] zur Datenverwaltung ihrer Dienste interessant macht. *Shared Process* stellt einen geeigneten Kompromiss aus Sicherheit, Performance und operativen Kosten dar und kommt aus diesem Grund beispielsweise bei Windows Azure SQL-Datenbank zum Einsatz.

Die dynamische Ressourcenverwaltung, Lastverteilung und Elastizität zählen zu den primären Anforderungen an mandantenfähige Datenbanksysteme [AJPK09, BZP+10]. Durch die verteilte Verarbeitung auf mehreren Rechnerknoten [Rah94] können sie eine Vielzahl von Mandanten adäquat unterstützen. Shared-Nothing-Architekturen weisen den Servern durch die Partitionierung des Datenbestands unabhängige Ressourcen zu, während die Server bei Shared-Disk-Architekturen einen zentralen Datenspeicher gemeinsam verwenden. Um auf Laständerungen adäquat reagieren zu können, bedürfen die Systeme geeigneter Platzierungs- und Migrationsstrategien [EDP+13, LHM+13, SJK+13, YQR+12] sowie

adäquater Migrationsverfahren [BCM+12, DNAEA11, EDAA11, SCM13] zur dynamischen Zuweisung der Mandanten zu Rechnerknoten. Dieser Beitrag konzentriert sich auf OLTP-Systeme, bei denen die Daten eines Mandanten nach Möglichkeit auf einen Server zu konzentrieren sind, um aufwändige verteilte Transaktionen zu vermeiden [CJZM10]. Migrationsstrategien legen zu migrierende Mandanten, Zielsysteme und Ausführungszeitpunkte der Migrationen fest. Sie überwachen den Ressourcenbedarf und die Performance von Mandanten und reagieren in Problemfällen, gegebenenfalls unter Einbeziehung von Lastvorhersagen auf Basis von Mandantenprofilen [CJP+11, EDP+13].

Im Rahmen dieses Beitrags zeigen wir vier Anforderungsprofile für die Live-Migration von Mandanten auf und ordnen ihnen Anforderungen an die Migrationsdurchführung zu. Wir kategorisieren Migrationsverfahren in drei Klassen und verdeutlichen deren Stärken und Schwächen. Zudem stellen wir das Framework KitMig zur flexiblen Live-Migration in relationalen Datenbanksystemen basierend auf dem Shared-Process-Ansatz vor. KitMig stellt verschiedene Module zur Verfügung, um den Migrationsablauf an vielfältige Anforderungen anzupassen und ist infolgedessen in unterschiedlichen Anwendungsgebieten verwendbar. Wir geben einen Überblick über die Implementierung KitMigs sowie über bereitgestellte Module und Verfahren. Gestützt durch die Ergebnisse verschiedener Untersuchungen leiten wir Einsatzbereiche ausgewählter KitMig-Verfahren ab.

Der Beitrag ist wie folgt aufgebaut: In *Kapitel 2* werden Bewertungskriterien und Anforderungsprofile von Migrationsverfahren aufgezeigt und Verfahrensklassen gegenübergestellt. *Kapitel 3* stellt das Framework KitMig vor und gibt einen Überblick über seine Implementierung und die bereitgestellten Module. *Kapitel 4* präsentiert die Ergebnisse verschiedener Untersuchungen. Es bewertet ausgewählte Migrationsverfahren und leitet ihre Eignung für die Anforderungsprofile ab. Im Rahmen des *Kapitels 5* werden veröffentlichte Verfahren zur Live-Migration vorgestellt und bewertet. Es geht zudem auf Forschungsgebiete mit vergleichbaren Anforderungen ein. *Kapitel 6* fasst die Ergebnisse des Beitrags zusammen und gibt einen Ausblick auf mögliche Erweiterungen und offene Fragestellungen.

2 Live-Migration

Als Migration wird das Übertragen eines Datenbestands von einem Quell- auf ein Zielsystem bezeichnet. Sie kann mit einem Wechsel der DBMS-Version oder des Betriebssystems verbunden sein und den Datenbestand etwa auf eine neue Festplatte innerhalb eines Servers, auf einen neuen Server innerhalb eines Rechenzentrums oder zu einem anderen Rechenzentrum übermitteln. Aufgrund ihrer großen Bedeutung in mandantenfähigen Datenbanksystemen ist sie nativ durch jene Systeme zu unterstützen [AJPK09, EDAA11].

Ein trivialer Migrationsansatz besteht in der Betriebsunterbrechung des Datenbanksystems und anschließender Erstellung, Übertragung und Wiederherstellung einer Sicherung. Ansätze wie dieses *Stop-And-Copy* [EDAEA11, SCM13] sind vor allem wegen nicht gewährleisteter Verfügbarkeit während der Migration für Cloud-Datenbankdienste nicht akzeptabel. Stattdessen sind Live-Migrationen gemäß *Definition 1* zu unterstützen.

Kategorie	Bewertungskriterium
Verfügbarkeit	Systemausfallzeit Ausfallzeit des Mandanten Fehlgeschlagene Mandantenanfragen Migration der Datenbankverbindung
Systembelastung	Mehrbelastung vor der Migration Mehrbelastung während der Migration Mehrbelastung nach der Migration Migrationsdauer Zeitpunkt der Lastmigration Kommunikationsaufwand
Koordination und Korrektheit	Autonomie Korrektheit Fehlertoleranz

Tabelle 1: Bewertungskriterien für Migrationsverfahren

► **Definition 1** *Als **Live-Migration** wird der Prozess innerhalb eines mandantenfähigen Datenbanksystems bezeichnet, der den gesamten Datenbestand eines Mandanten von einem Quell- auf ein Zielsystem übermittelt und dabei folgenden Kriterien genügt [EDAA11]:*

- *Keine Verursachung von Ausfallzeiten des Datenbanksystems,*
- *keine oder sehr geringe Beeinflussung der Erreichbarkeit der Daten des Mandanten,*
- *Verursachung keiner oder nur sehr weniger Transaktionsabbrüche des Mandanten,*
- *Übertragung bestehender DB-Verbindungen zum Zielsystem ohne Neuaufbau.*

2.1 Bewertungskriterien

Geeignete Metriken ermöglichen die Vergleichbarkeit verschiedener Migrationsverfahren. Sie werden in *Tabelle 1* aufgelistet, wobei die Kategorie Verfügbarkeit alle wesentlichen Merkmale einer Live-Migration gemäß *Definition 1* beinhaltet. Die weiteren Bewertungskriterien werden im Folgenden nach [BZP+10, EDAA11, EDAEA11] beschrieben.

Live-Migrationen führen insbesondere durch den Ressourcenbedarf ihrer Datenbewegungen unweigerlich zu einer Belastung des Quellsystems, Zielsystems und Netzwerks. Mehrbelastungen können sich als Erhöhung der Antwortzeit oder als Verringerung des Transaktionsdurchsatzes auswirken und sind für den zu migrierenden Mandanten und für zeitgleich aktive Mandanten messbar. Sie können bestehende Überlastungen verschärfen oder gar neue hervorrufen [BCM+12]. Notwendige Vorgänge im operativen Betrieb, welche die Voraussetzungen für Live-Migrationen schaffen, führen zu einer Systembelastung vor der Migration. Mehrbelastungen können zudem während der Migration auftreten. Der Verzicht auf das Übertragen des Pufferstatus von Datenbankseiten des migrierten Mandanten

(im Folgenden als Pufferinhalt bezeichnet) ist ein typisches Beispiel für eine Mehrbelastung, die aus einer Migration resultiert. Wichtige Einflussfaktoren auf die Systembelastung sind zudem die Migrationsdauer, der Übertragungszeitpunkt der Transaktionsverantwortung an das Zielsystem (Lastmigration) und der Kommunikationsumfang, der sowohl die übermittelten Mandantendaten als auch zusätzlich benötigte Inhalte umfasst.

Spezielle DBMS-Komponenten auf dem Quell- und Zielsystem sind für die Migrationsdurchführung und die korrekte Datenübertragung verantwortlich. Die Autonomie kennzeichnet den nötigen Koordinierungsaufwand zur Veranlassung und Durchführung der Migration. Während Korrektheit bei vollständiger Datenübertragung an das Zielsystem und ordnungsgemäßer Integration der Daten im Zielsystem gegeben ist, bewertet die Fehlertoleranz die transaktionale Korrektheit bei Ausfällen und in Fehlersituationen.

Nicht alle aufgeführten Metriken können von einem Migrationsverfahren gleichermaßen gut erfüllt werden. So steht beispielsweise eine geringe Systembelastung während der Migration in der Regel im Widerspruch zu einer geringen Migrationsdauer. Der Stellenwert jeder Metrik für eine Migration wird wesentlich durch das zugrunde liegende Anforderungsprofil bestimmt.

2.2 Anforderungsprofile

Die Live-Migration ist eine Schlüsselkomponente zur Optimierung der Ressourcenauslastung, indem sie die Zuweisung von Systemressourcen an Mandanten anpasst. Sie dient der Lastverteilungen, der Konsolidierung von Servern mit anhaltend geringer Auslastung (Serverkonsolidierung, Scale-In) und der Nutzung von Ressourcen hinzugefügter Server (Scale-Out). Lastverteilungen und Scale-Outs realisieren in der Regel Entlastungen von Quellsystemen mit (zu) hoher Last auf Zielsysteme mit geringer oder keinerlei Last, während ein Scale-In in der Regel bei (zu) geringer Last des Quellsystems durchgeführt wird und überlastete Systeme als Ziel meidet. Diese Unterscheidung führt in Verbindung mit den Ursachen einer Migration und der Dringlichkeit des Eingreifens zu den im Folgenden beschriebenen Anforderungsprofilen an Live-Migrationen.

Akute Entlastung: Diese Migrationen dienen der unverzüglichen Entlastung des Quellsystems und sind durch bestehende oder unmittelbar drohende finanzielle Aufwendungen für den Datenbankbetreiber aufgrund von Missständen auf dem Quellsystem motiviert. Zu den Hauptbeweggründen zählen Überlastungen des Quellsystems und Verletzungen der Service-Level-Agreements (SLAs) von Mandanten infolge von unvorhersehbarem Nutzungsverhalten. Zur schnellen Entlastung des Quellsystems ist eine möglichst frühe Lastmigration vonnöten. Zudem sind Mehrbelastungen auf dem Quellsystem und aus Migrationen resultierende Mehrbelastungen zu minimieren, um die Situation nicht weiter zu verschärfen.

Vorausplanende Entlastung: Dieses Profil umfasst Migrationen mit dem Ziel der Entlastung des Quellsystems zur Vermeidung zukünftiger Überlastungen. Zu ihren Ursachen zählen beispielsweise eine serverübergreifende Lastbalancierung, SLA-Änderungen oder erwartete SLA-Verletzungen. Zudem können sie durch erwartete Last-

anstiege, proaktiv hinzugefügte Server oder eine höhere Anzahl zu verwaltender Mandanten hervorgerufen werden. Sie bedürfen einer frühen Lastmigration sowie einer geringen Mehrbelastung auf dem Quellsystem und nach der Migration.

Akute Serverkonsolidierung: Unaufschiebbare Wartungsarbeiten stellen die Hauptursache für eine umgehende Deaktivierung von Datenbankservern mit dem Ziel einer akuten Serverkonsolidierung dar. Sie werden beispielsweise durch Fehlermeldungen sich selbst überwachender Hardware ausgelöst und bedürfen der umgehenden Migration aller durch den Server verwalteten Mandanten. Die Migrationen sollten eine minimale Laufzeit anstreben und Mehrbelastungen auf dem Zielsystem minimieren, um dessen Überlastung zu vermeiden. Mehrbelastungen auf dem Quellsystem und eine späte Lastmigration können hingegen in Kauf genommen werden.

Vorausplanende Serverkonsolidierung: Migrationen dieses Profils führen zeitunkritische Serverkonsolidierungen durch. Mögliche Ursachen bestehen in geplanten Wartungsarbeiten oder der Entfernung von Servern mit geringer Last aus Kostengründen. Eine reduzierte Last kann beispielsweise durch zurückgehende Ressourcenanforderungen der Mandanten oder einer Reduzierung der Mandantenanzahl hervorgerufen werden. Analog zu akuten Serverkonsolidierungen sind hierbei Mehrbelastungen für das Zielsystem zu vermeiden. Eine relativ lange Migrationsdauer ist bei entsprechendem Planungshorizont hingegen tolerierbar.

2.3 Verfahrensklassen

Verfahren zur Live-Migration in mandantenfähigen Datenbanksystemen müssen folgende Kernprozesse realisieren:

Datenmigration: Dieser Prozess ist für die Übertragung des kompletten Datenbestands des zu migrierenden Mandanten vom Quell- auf das Zielsystem verantwortlich. Die Daten können durch das Quellsystem bereitgestellt (Push-Strategie) oder durch das Zielsystem bedarfsgerecht angefordert (Pull-Strategie) werden. In Shared-Disk-Architekturen sind ausschließlich Arbeitsspeicherinhalte wie Pufferinhalte des Mandanten zu übertragen, während Shared-Nothing-Architekturen die Übertragung persistenter Daten erfordern und die Migration der Pufferinhalte optional ist.

Lastmigration: Sie leitet bestehende Datenbankverbindungen des Mandanten vom Quellsystem auf das Zielsystem um. Laufende Transaktionen auf dem Quellsystem sind abzubrechen, ebenfalls zu migrieren oder weiterhin auf dem Quellsystem auszuführen und mit neuen Transaktionen auf dem Zielsystem zu synchronisieren.

Bereinigung: Nach erfolgter Daten- und Lastmigration ist der Datenbestand des migrierten Mandanten auf dem Quellsystem vollständig zu entfernen.

Migrationsverfahren variieren bezüglich der Prozessabfolge und -implementierung. Dabei ist zwischen reaktiven (post-copy), proaktiven (pre-copy) und hybriden Verfahren zu unterscheiden [AEDE11, SCM13].

Reaktive Verfahren [EDAEA11] stellen dem Zielsystem zu Migrationsbeginn die zur Verarbeitung von Transaktionen des Mandanten notwendigen Kataloginhalte bereit. Anschließend führen sie die Lastmigration durch, woraufhin das Zielsystem im Rahmen der Datenmigration die zur Transaktionsverarbeitung notwendigen Daten vom Quellsystem anfordert. Dieser Vorgang wird durch parallele Bereitstellung der restlichen Daten durch das Quellsystem begleitet. Die Verfahren erreichen eine schnelle Entlastung des Quellsystems. Die bedarfsgerechte Anforderung führt jedoch zu einem großem Kommunikationsaufwand, einer langen Migrationsdauer und einer hohen Belastung während der Migration. Reaktive Verfahren eignen sich daher insbesondere für die akute und vorausplanende Entlastung.

In *proaktiven Verfahren* [BCM+12, DNAA10, DNAEA11] stellt das Quellsystem dem Zielsystem bereits vor der Lastmigration den kompletten Datenbestand bereit. Durch die gebündelte Übertragung des Datenbestands halten sie den Kommunikationsaufwand und die Mehrbelastungen während der Migration gering und reduzieren die Migrationsdauer. Jedoch entlasten sie durch die lange Datenübertragungsdauer vor der Lastmigration das Quellsystem sehr spät. Demzufolge eignen sich proaktive Verfahren vorrangig bei akuten und geplanten Serverkonsolidierungen.

Hybride Verfahren [SCM13] verbinden die vorherigen Ansätze durch die verschränkte Ausführung der Daten- und Lastmigration sowie die Kombination von Datenbereitstellungen und -anforderungen. Dies erlaubt insbesondere die Bereitstellung häufig verwendeter Mandantendaten (Hot Pages) vor der Lastmigration, während nach ihr die weiteren Daten (Cold Pages) bereitgestellt oder bedarfsgerecht angefordert werden. Die Bereitstellung von Hot Pages dient der Reduzierung bedarfsgerechter Seitenanforderungen. Der Anteil der Hot Pages am Gesamtdatenbestand bestimmt die Charakteristik der Migration, welche sich zwischen proaktiven Verfahren (ausschließlich Hot Pages) und reaktiven Verfahren (keine Hot Pages) bewegt.

Die Bewertung der Verfahrensklassen und die Ableitung ihrer Eignung für die verschiedenartigen Anforderungsprofile von Live-Migrationen unterstreichen, dass kein universelles Migrationsverfahren existieren kann, das in allen Anwendungsgebieten uneingeschränkt verwendbar ist. Hybride Verfahren stellen einen geeigneten Ansatzpunkt hierzu dar, sie müssen jedoch Freiheitsgrade zur Abstimmung ihres Ablaufes auf den jeweiligen Anwendungsfall bereitstellen.

3 KitMig

Mit der Entwicklung von KitMig verfolgten wir das Ziel, verschiedene Migrationsabläufe in einem Migrationsverfahren für mandantenfähige Datenbanksysteme zu vereinen. Diese Flexibilität erlaubt dem DBMS die Festlegung des geeigneten Migrationsablaufs in Abhängigkeit vom gegebenen Anforderungsprofil und dem Systemzustand. Die folgenden Abschnitte erläutern die Systemarchitektur, stellen die Migrationsphasen und zugehörige Module KitMigs vor und gehen auf die Implementierung und bestehende Einschränkungen ein. Eine umfassende Beschreibung des Frameworks ist in [Göb15] zu finden.

3.1 Systemarchitektur und Design-Entscheidungen

KitMig setzt einen Datenbank-Cluster mit Shared-Nothing-Architektur voraus. Jeder Datenbankserver verwaltet dabei die Daten mehrerer Mandanten, wobei KitMig die maximale Anzahl der Mandanten je Server nicht einschränkt. Gemäß dem Konsolidierungsansatz *Shared Process* wird jedem Mandanten ein Datenbankschema zur Speicherung seiner Datenbankobjekte zugewiesen. Im Vergleich zu *Shared Machine* ruft dieser Ansatz zusätzlichen Herausforderungen für Live-Migrationen wie die gemeinsame Nutzung des Datenbankpuffers und der Datenbankdateien sowie potenzielle Konflikte der Seiten-IDs bei ihrer Einbindung auf dem Zielsystem hervor, mit denen KitMig umgeht. Der Einsatz KitMigs für *Shared Machine* bedarf nur geringfügiger Anpassungen. Der Ansatz *Shared Table* bringt hingegen weitere Herausforderungen mit sich, so schließt beispielsweise die Speicherung der Daten mehrerer Mandanten innerhalb einer Datenbankseite den Einsatz KitMigs aus. Aufgrund seiner Konzipierung für OLTP-Systeme unterstellt KitMig Transaktionen mit kurzer Laufzeit und wenigen Zugriffen auf Datenbankseiten. Bereichsanfragen wirken sich in KitMig negativ auf die Mehrbelastung während der Migration, den Kommunikationsaufwand und die Migrationsdauer aus. KitMig setzt zudem voraus, dass die Ressourcen eines Datenbankservers zur adäquaten Verwaltung aller Daten eines Mandanten genügen und keine knotenübergreifende Partitionierung angewendet wird, was einen gebräuchlichen Ansatz [YSY09] darstellt.

Unter den gegebenen Voraussetzungen umfasst die Live-Migration eines Mandanten bei KitMig die Übertragung aller Inhalte eines Datenbankschemas von einem Quell- an einen Zielserver. Ein Server kann als Quelle und Ziel mehrerer paralleler Migrationen agieren. Die Migration wird über eine SQL-Erweiterung durch Administratoren oder interne DBMS-Komponenten wie Controller zur Realisierung von Migrationsstrategien ausgelöst. Die Lastmigration realisiert KitMig durch einen externen Proxy-Server, der horizontale Skalierbarkeit bietet und eingehende Anfragen basierend auf dem Standardschema der Datenbankverbindung an den zuständigen Datenbankserver umleitet. KitMig passt die Umleitungsvorschriften im Rahmen einer Migration an.

3.2 Migrationsphasen

Abbildung 1 verdeutlicht den Migrationsablauf bei KitMig, der in vier Phasen erfolgt: Datenbereitstellung durch den Quellserver, Synchronisierung paralleler Transaktionsverarbeitung (Dualbetrieb), Datenanforderung durch den Zielserver und Bereinigung des Quellservers. Aufgrund seines Ablaufs ist KitMig als hybrides Migrationsverfahren einzuordnen. Es folgt eine kurze Erläuterung der Funktionen aller Migrationsphasen.

Die *Bereitstellungsphase* erlaubt dem Zielserver die Verarbeitung von Transaktionen in zukünftigen Phasen. Dies wird durch die Übertragung eines Schema-Grundgerüsts ermöglicht, welches die Schema- und Datendefinitionen zu Tabellen, Indizes, Views, Sequenzen etc. sowie die Indexstrukturen ohne jegliche Mandantendaten enthält. Zudem erlaubt KitMig die Bereitstellung einer beliebige Menge der im Puffer befindlichen Mandantendaten.

Die *Synchronisationsphase* dient zur Lastmigration und Abarbeitung der am Quellserver

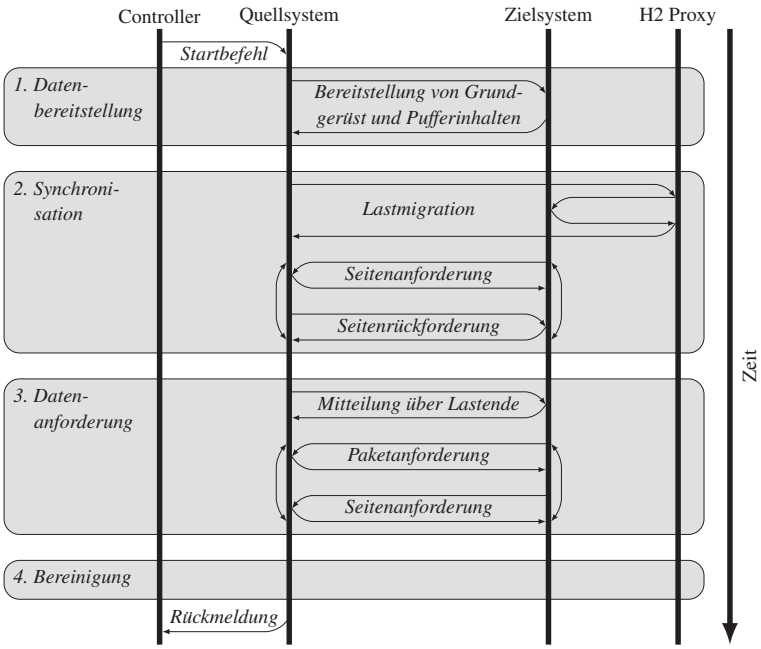


Abbildung 1: Ablauf einer Migration mit KitMig

noch ausstehenden Transaktionen, während neu eingehende Anfragen bereits durch den Proxy zum Zielserver umgeleitet werden. Der exklusive Datenzugriff zwischen den Servern wird durch ein Eigentumskonzept der Mandantendaten geregelt. KitMig ordnet Datenbankseiten einen Eigentümer zu und ermöglicht den Änderung des Eigentümers durch die Anforderung und Rückforderung des Seiteneigentums und -inhalts.

Die *Anforderungsphase* beginnt nach der Verarbeitung der letzten am Quellserver aktiven Transaktion. Zu diesem Zeitpunkt liegen die Mandantendaten teilweise bereits am Zielserver vor. Er fordert alle im Besitz des Quellservers befindlichen Datenbankseiten an. Für laufende Transaktionen benötigte Daten werden analog zur Synchronisationsphase seitenweise angefordert, während die weiteren Daten in Paketen übertragen werden.

Die *Bereinigungsphase* dient der abschließenden Entfernung des migrierten Datenbankschemas am Quellserver und der Migrationsmetadaten auf beiden Servern.

3.3 Vergleich bereitgestellter Module

KitMig stellt für die Phasen 1 – 3 verschiedene Module und Parameter zur individuellen Anpassung der Migrationsdurchführung an das Anforderungsprofil bereit. Dieser Abschnitt stellt die Module jeder Phase gegenüber. Das Modul `PUSH_WIREFRAME` stellt dem Zielserver das Schema-Grundgerüst bereit. Es realisiert ein reaktives Verfahren zur schnellen Entlastung des Quellservers, ist jedoch mit langen Zugriffszeiten aufgrund von vielen

Modul in	Eigentumsrecht		Seiteninhalt	
Synchronisationsphase	**QS ⇒ ZS**	**QS ⇐ ZS**	**QS ⇒ ZS**	**QS ⇐ ZS**
SIMPLEX	+	–	+	–
DUPLEX_UNMODIFIED	+	+/–	+	–
DUPLEX_ALL	+	+	+	+

QS: Quellserver ZS: Zielserver +: Zulässiger Transfer –: Unzulässiger Transfer

Tabelle 2: Gegenüberstellung der Übertragungsvorschriften zur Synchronisation

Seitenanforderungen in den Folgephasen verbunden. Im Vergleich dazu erzielt PUSH_BUFFER durch das Übertragen von Pufferinhalten kürzere Zugriffszeiten auf die meistverwendeten Mandantendaten auf Kosten einer späteren Lastmigration. Die Lastsituation auf dem Quellserver und der Aktivitätsgrad des Mandanten bestimmen hierbei die Charakteristik der Migration, welche sich zwischen proaktiven Verfahren (vollständige Pufferung der Mandantendaten) und reaktiven Verfahren (keine Pufferinhalte des Mandanten) bewegt.

Die Module der Synchronisationsphase ermöglichen die Abwägung zwischen einer Vielzahl abgebrochener Transaktionen und hohem Kommunikationsaufwand. *Tabelle 2* verdeutlicht, dass alle Module die Übertragung von Eigentumsrechten und Seiteninhalten zum Zielserver gewähren. Das Modul SIMPLEX erlaubt ausschließlich die genau einmalige Übertragung von Daten zum Zielserver und bietet somit die geringste Netzwerkbelastung. Transaktionen auf dem Quellserver, die Zugriff auf bereits übertragene Daten benötigen, müssen jedoch abgebrochen werden. DUPLEX_UNMODIFIED erlaubt die Übertragung des Eigentumsrechts unveränderter Daten zurück zum Quellserver. Das liberalste Modul DUPLEX_ALL bringt dagegen keine Einschränkung von Seitenübertragungen mit sich und verursacht somit keine Transaktionsabbrüche aufgrund von Eigentumskonflikten.

KitMig stellt zwei Module für die Anforderungsphase bereit. Das Modul PULL_AND_BUFFER dient der Erwärmung des Datenbankpuffers auf dem Zielserver. Im Rahmen von Seitenpaketen übermittelte Datenbankseiten fügt er in den Datenbankpuffer ein, falls sie sich im Datenbankpuffer des Quellservers befanden. Als Beispielszenario sei hier die akute Entlastung eines sehr aktiven Mandanten genannt, die eine schnelle Entlastung des Quellservers fordert und daher unter Umständen nicht alle zu migrierenden Pufferdaten bereits vor der Lastmigration übertragen kann. Alternativ kann PULL_ONLY gewählt werden, um die verbleibenden Daten ohne Übernahme des Pufferzustands zu migrieren.

3.4 Implementierung

KitMig wurde prototypisch in das Open-Source-DBMS H2 (Version 1.3.171) integriert, da es alle benötigten Funktionen wie Isolation auf Schema-Ebene, Multithreaded-Sitzungen und baumbasierte Primär- und Suchindextypen mitbringt. Zudem bietet es einen übersichtlich strukturierten und zugleich überschaubaren Programmcode. Lediglich die auf Tabel-

lenebene erfolgende Sperrverwaltung in H2 führte zu Herausforderungen bei der Realisierung KitMigs. Dessen Konzeption ist nicht auf H2 zugeschnitten, sondern lässt eine Integration in andere DBMS zu.

Der SQL-Befehlssatz von H2 wurde um Befehle für die Initiierung der Migration (inklusive der Module und Parameterwerte), die Anpassung des Ablaufs laufender Migrationen und die Server-zu-Server-Kommunikation erweitert. Am Quellserver wird der Migrationsablauf durch einen Controller-Thread koordiniert. Zudem verfügt jede Datenbank über einen Migrations-Manager, der Metadaten wie Authentifizierungsdaten und Serveradressen zu laufenden Migrationen verwaltet. Um die Unveränderlichkeit des Schema-Grundgerüsts zu erzwingen, setzt KitMig zu Beginn der Migration eine Strukturspere, die Veränderungen an den Datendefinitionen und Indexstrukturen während der Initialisierungs- und Synchronisationsphasen verbietet. Sie unterbindet die Ausführung von DDL-Anweisungen zur Erstellung, Änderung und Entfernung von Datenbankobjekten des Schemas sowie DML-Anweisungen, die zu Änderungen der Indexstrukturen führen. Dies betrifft einen Teil der `UPDATE`-Befehle sowie alle `INSERT`- und `DELETE`-Befehle.

Anschließend erstellt KitMig das Schema-Grundgerüst durch die Serialisierung der Suchindizes sowie der inneren Knoten der Primärindizes. Beim Modul `PUSH_BUFFER` werden zusätzlich die im Datenbankpuffer des Quellservers befindlichen Seiten des Mandanten in das Schema-Grundgerüst integriert. Zur Verhinderung einer späten Lastmigration sehr aktiver Mandanten erlaubt KitMig die Einschränkung der zu migrierenden Pufferinhalte durch einen Parameter. Auf die Erzeugung dieses Grundgerüsts folgt seine Übertragung zum Zielserver als LOB-Parameter innerhalb eines SQL-Befehls. Die Daten werden auf dem Zielserver deserialisiert und in die lokale Datenbank eingebunden. Bei der Deserialisierung von Seitendaten wird zur Vorbeugung potenzieller Konflikte der Seiten-IDs auf dem Quell- und Zielserver jeder einzubindenden Seite eine neue ID zugeteilt und die Zuordnung der IDs des Quell- und Zielservers für die spätere Kommunikation zwischen den Servern aufbewahrt. Die Strukturspere gewährleistet die Konsistenz dieser Zuordnung während der Migration.

Das Eigentumskonzept ermöglicht exklusive Seitenzugriffe auf Blattknoten der Primärindizes während der Migration. Der Zielserver erhält zu Beginn der Synchronisationsphase das Eigentum an den migrierten Pufferinhalten, während alle weiteren Blattknoten dem Quellserver zugewiesen werden. Seitenzugriffe auf dem Quell- und Zielserver sind stets mit einer Vorabprüfung des Seiteneigentums verbunden. Besitzt der Server die Seite nicht, so ist sie vom anderen Server anzufordern. Das verwendete Modul der Synchronisationsphase entscheidet über die Zulassung der Eigentumsübertragung. Nicht zulässige Seitenanforderungen führen zum Abbruch der zugehörigen Transaktion. Zur Realisierung des Moduls `DUPLEX_UNMODIFIED` überwacht KitMig Änderungen übertragener Datenbankseiten. Diese Überwachung erlaubt KitMig zudem die Vermeidung unnötiger Übertragungen von Seiteninhalten, indem es bei Anforderungen unveränderter Datenbankseiten lediglich das Eigentumsrecht überträgt und auf die Übertragung des Seiteninhalts verzichtet. Seiteninhalte sind folglich nur nach Änderungen wiederholt zu übertragen, was minimalen Netzwerk-Overhead bei lesenden Anfragen bedeutet. Während der Synchronisationsphase bestimmt KitMig regelmäßig die Anzahl der auf dem Schema aktiven Sitzungen. Sobald alle Sitzungen geschlossen wurden, endet die Synchronisationsphase und

der Quellserver signalisiert dem Zielserver den gemeinsamen Wechsel in die Anforderungsphase. Durch einen optionalen Parameter kann die Dauer der aufwendigen Synchronisationsphase begrenzt werden, indem nach Ablauf der maximalen Laufzeit langlaufende Transaktionen gezielt abgebrochen werden.

In der Anforderungsphase ermittelt der Zielserver für alle Tabellen die noch zu übertragenden Datenbankseiten und fordert sie in Paketform an. Die Übertragung und Einbindung der Seiten beeinflusst sowohl das Netzwerk als auch die Transaktionsverarbeitung auf dem Zielserver. Zur Regulierung der Belastung sind zu Migrationsbeginn die Größe der Pakete und die Wartezeit zwischen den Paketanforderungen festzulegen. KitMig erlaubt zudem die Anpassung beider Werte während des Migrationsverlaufs und eine dynamische Drosselung der zu verwendenden Netzwerkbandbreite für die Paketübertragung, um die Mehrbelastungen in dieser Phase zu kontrollieren.

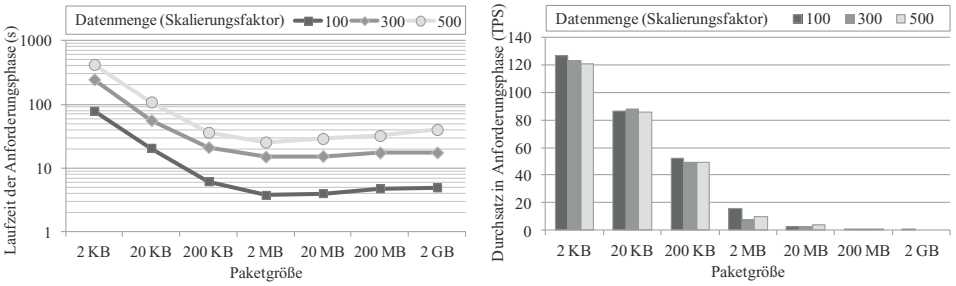
3.5 Einschränkungen

KitMig erlaubt die Migration persistenter Standard-Indextypen. Nicht unterstützt wird die Migration von reinen In-Memory-Indizes und Large Objects. Die Daten eines Mandanten sind zudem vollständig zu übertragen, eine Aus- oder Abwahl einzelner Tabellen ist nicht möglich. KitMig protokolliert alle Informationen auf dem Quell- und Zielserver, die zur Wiederherstellung des Systemzustands und Migrationsstatus nach einem Systemausfall auf einem oder beiden Servern notwendig sind. Die angelegten Dateien beinhalten den Migrationsablauf, die Zugangsdaten der beteiligten Server, die aktuelle Migrationsphase, das Mapping von Seiten-IDs und die Eigentumsrechte. Die Live-Migration verursacht zusätzliche Herausforderungen für eine log-basierte Systemwiederherstellung, deren Realisierung zu unseren zukünftigen Arbeiten gehört.

4 Experimentelle Untersuchungen

Dieses Kapitel stellt nach einem Überblick der Testumgebung die Ergebnisse zweier ausgewählter Untersuchungen des KitMig-Frameworks vor. Bei den Untersuchungen kamen vier baugleiche Server zum Einsatz, die über 1 Gigabit Ethernet miteinander verbunden sind. Sie besitzen jeweils eine 3.4 GHz Intel Core i7-2600 CPU (4 Kerne), 8 GB RAM, eine Festplatte mit 7200 RPM und werden durch Debian 4.7.2-4 (64 Bit) betrieben. Zwei Server agierten als Datenbankserver mit einem um KitMig erweiterten H2 basierend auf der H2-Version 1.3.171. Jeder Datenbankserver verwaltet die Daten mehrerer Mandanten in einer gemeinsamen Datenbank durch Zuweisung separater Schemata. Der im Rahmen des Projekts entwickelte Datenbank-Proxy *H2 Proxy* [Göb14a] wurde auf einem separaten Proxy-Server betrieben. Er nimmt alle Anweisungen entgegen und leitet sie gemäß dem verwendeten Schema an den zuständigen Datenbankserver weiter (siehe Abschnitt 3.1).

Der vierte Server diente der Daten- und Lastgenerierung durch das für mandantenfähige Datenbanksysteme entwickelte Benchmark-Framework *MuTeBench* [Göb14b]. Wir verwendeten es für Untersuchungen basierend auf dem Yahoo! Cloud Serving Benchmark



(a) Laufzeit der Anforderungsphase

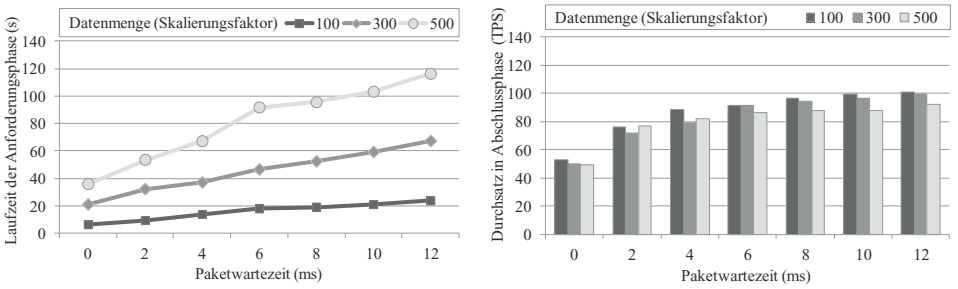
(b) Durchsatz in der Anforderungsphase

Abbildung 2: Einfluss der Paketgröße auf die Anforderungsphase (ohne Paketwartezeit)

(YCSB) [CST+10]. Im Gegensatz zu klassischen Datenbank-Benchmarks wurde dieser zur Evaluation von Cloud-Datenbanken entwickelt. Hierzu stellt er mehrere Lastszenarien mit variierendem Anteil lesender Anfragen bereit und bietet unterschiedliche Zugriffsmuster wie Gleichverteilung oder Zipf-Verteilung. Wir erweiterten YCSB um neue Lastszenarien und verwendeten in den Untersuchungen einen Transaktionsmix mit 70 % `READ`, 10 % `UPDATE`, 10 % `INSERT` und 10 % `DELETE` bei jeweils 10 Anweisungen pro Transaktion und Verwendung der Zipf-Verteilung. Jeder Mandant besitzt genau eine Tabelle mit 11 Spalten und 100.000 Zeilen (YCSB-Skalierungsfaktor 100, ca. 200 MB), 300.000 Zeilen (Faktor 300, ca. 610 MB) oder 500.000 Zeilen (Faktor 500, ca. 990 MB). Wir verwendeten eine Seitengröße von 2 KB, eine Puffergröße von 5 GB und das Isolationslevel Read Committed.

4.1 Paketgröße und -wartezeit

Als Grundlage für den Vergleich von KitMig-Verfahren sei auf den Einfluss verschiedener Parameter auf die Anforderungsphase eingegangen. Der effizienten Datenübertragung kommt in dieser Phase eine besondere Bedeutung zu, da sie insbesondere bei der Wahl von `PUSH_WIREFRAME` oder bei einem geringen Anteil der bereitgestellten Pufferinhalte an den zu migrierenden Daten den Großteil der Migrationsdauer ausmacht. Die Mandantendaten wurden mittels einer H2-Funktion komplett in den Puffer geladen, um Nebeneffekte aufgrund von Festspeicherzugriffen zu reduzieren. Es kam das KitMig-Verfahren `PUSH_WIREFRAME`, `SIMPLEX` und `PULL_ONLY` zum Einsatz. Beim ersten Test verzichteten wir auf Paketwartezeiten und nutzten Paketgrößen von einer Seite (2 KB) bis zu einer Million Seiten (2 GB). Mit zunehmender Paketgröße verringert sich der Kommunikationsaufwand pro übertragener Seite. Zudem kann die Effizienz der Seitenerfassung auf dem Quellserver und der Seitenintegration auf dem Zielserver erhöht werden und der Zielserver fordert weniger Seiten zur Einzelübertragung an. *Abbildung 2a* zeigt die resultierende Verringerung der Laufzeit der Anforderungsphase. Ab Paketgrößen von 20 MB wird nicht mehr von paralleler Serialisierung und Paketübermittlung profitiert, weshalb die



(a) Laufzeit der Anforderungsphase

(b) Durchsatz in der Anforderungsphase

Abbildung 3: Einfluss der Paketwartezeit auf die Anforderungsphase (Paketgröße: 200 KB)

Laufzeiten ansteigen. Sperren bei der Integration schränken die Verfügbarkeit der Daten während der Deserialisierung ein. *Abbildung 2b* zeigt, dass durch die längere Sperrdauer bei zunehmender Paketgröße folgerichtig die Transaktionsverarbeitung auf dem Zielserver deutlich beeinflusst oder gar verhindert wird.

Die Paketgröße von 200 KB stellt einen Kompromiss aus passabler Laufzeit und hohem Durchsatz dar. Wir verwendeten sie bei der Untersuchung von Paketwartezeiten. Der Zielserver wartete in den Untersuchungen bis zu 12 Millisekunden zwischen der Deserialisierung der Seiten eines Pakets und der Anforderung des folgenden Pakets. *Abbildung 3a* verdeutlicht den nahezu konstanten Anstieg der Laufzeit bei zunehmender Wartezeit. Gemäß *Abbildung 3b* ist bis 6 ms Wartezeit ein deutlicher Anstieg des Durchsatzes erkennbar. Daher kann mit 200 KB-Paketen und 4 ms Wartezeit bei deutlich geringerer Laufzeit ein ähnlicher Durchsatz erzielt werden wie bei einer Paketgröße von 20 KB ohne Wartezeit. Die *Abbildungen 2b* und *3b* verdeutlichen, dass der erzielte Durchsatz in beiden Untersuchungen durch größere Datenmengen nur geringfügig beeinflusst wird, was insbesondere auf die Pufferung der gesamten Datenmenge zurückzuführen ist.

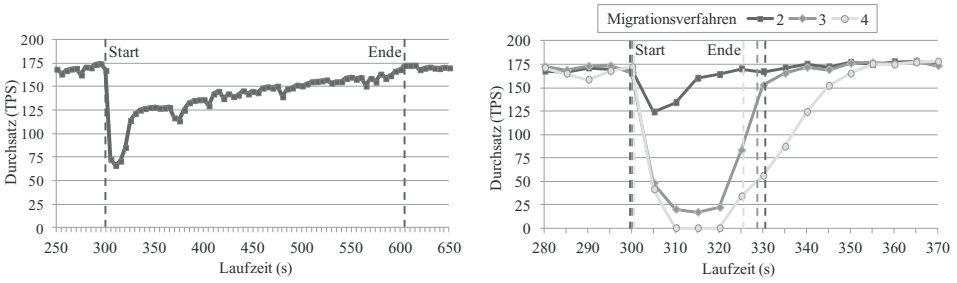
Wir führten zudem Untersuchungen mit statischer und dynamischer Drosselung der Übertragungsgeschwindigkeit durch. Sie ist ein geeignetes Mittel zur Reduzierung der Systemlast. Ihre regelmäßige Kontrolle der zu übertragenen Datenmenge bedingt jedoch einen erhöhten Aufwand, weshalb wir uns für die folgende Untersuchung auf feste Wartezeiten zwischen dem Versand von Paketen beschränkten.

4.2 Gegenüberstellung ausgewählter Verfahren

Aufgrund der Vielzahl möglicher Kombinationen von Modulen und Parameterwerten ist eine Untersuchung aller denkbaren Verfahren nicht realisierbar. Zur Gegenüberstellung von KitMig-Verfahren wählten wir gezielt vier Verfahren aus, deren Module und Parameterwerte in *Tabelle 3* zusammengefasst sind. Die Verfahren zeigen die Bandbreite der Möglichkeiten KitMigs auf. Die Daten des zu migrierenden Mandanten (Skalierungsfaktor 300, ca. 610 MB) wurden bei dieser Untersuchung nicht vollständig gepuffert, der Quellserver jedoch zur Puffererwärmung vor der Migration einer 5-minütigen Last ausgesetzt.

Nr.	Modul in Bereitstellungsphase	Modul in Synchronisationsphase	Modul in Anforderungsphase	Paket-größe	Paket-wartezeit
1	PUSH_WIREFRAME	DUPLEX_UNMODIFIED	PULL_AND_BUFFER	20 KB	4 ms
2	PUSH_BUFFER	DUPLEX_ALL	PULL_ONLY	200 KB	4 ms
3	PUSH_WIREFRAME	SIMPLEX	PULL_AND_BUFFER	2 MB	2 ms
4	PUSH_WIREFRAME	SIMPLEX	PULL_ONLY	2 GB	0 ms

Tabelle 3: Module und Parameterwerte der untersuchten KitMig-Verfahren



(a) Durchsatz für Verfahren 1

(b) Durchsatz für Verfahren 2 – 4

Abbildung 4: Erzielter Durchsatz der untersuchten KitMig-Verfahren

Verfahren 1 verfolgt das Ziel einer geringen Quellserverbelastung und ist insbesondere für die vorausplanende Entlastung und mit Einschränkungen für die akute Entlastung geeignet. Nach der Übertragung des Schema-Grundgerüsts mit lediglich 74 abgebrochenen Transaktionen (Abbruchquote von <0,1 % über den gesamten Migrationszeitraum) kann hierbei die Transaktionsverantwortung im Rahmen einer kurzen Synchronisationsphase schnell auf den Zielserver übertragen werden. Bei der Festlegung der Paketgröße und Paketwartezeit wurde das Ziel einer geringen Belastung während der Migration verfolgt. Die gewählten Werte führen zu einer ausgedehnten Anforderungsphase (siehe *Abbildung 5*), in der der Quellserver die Daten zur Entlastung nur langsam an den Zielserver übermittelt. In der Folge werden beträchtliche 24 % der Seiten vom Zielserver nach Bedarf angefordert und der Durchsatz des migrierten Mandanten nähert sich nur langsam dem Optimum an (siehe *Abbildung 4a*). Bei Migrationen von Mandanten mit Performance-Problemen sind daher andere Parameterwerte zu wählen.

Verfahren 2 überträgt in der Bereitstellungsphase die kompletten Pufferinhalte des Mandanten (58 % des Gesamtvolumens) zum Zielserver. In der aktuellen KitMig-Version bedarf es hierzu einer Strukturspere, die zu 525 Transaktionsabbrüchen (Abbruchquote von ca. 1,4 % über den gesamten Migrationszeitraum) führt und die Verfügbarkeit somit in Form von Dienstunterbrechungen einschränkt. Nach einer kurzen Synchronisationsphase werden die verbliebenen Daten angefordert. Im Vergleich zu Verfahren 1 sorgen sowohl die erhöhte Paketgröße als auch das erheblich verringerte Datenvolumen der anzufordernden Daten zu einer deutlich schnelleren Übertragung verbliebener Seiten (siehe *Abbildung 4b*). Der Zielserver fordert daher nur 6 % der Seiten nach Bedarf an. Das Verfahren eignet

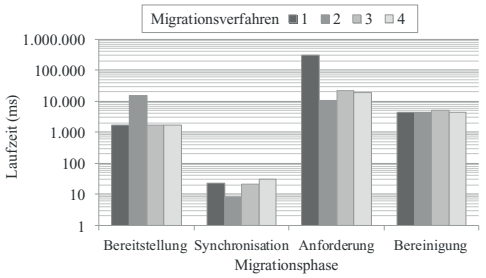


Abbildung 5: Laufzeit der untersuchten KitMig-Verfahren

sich für vorausplanende Serverkonsolidierungen und ist mit Einschränkungen auch bei der akuten Serverkonsolidierung einsetzbar.

Verfahren 3 besitzt das Ziel einer kurzen Migrationsdauer bei geringer Dienstunterbrechung und bietet sich folglich für akute Serverkonsolidierungen an. Dazu überträgt es das Schema-Grundgerüst an den Zielserver und verfolgt bei der gewählten Paketgröße und -wartezeit das Ziel einer schnellen Übertragung der Mandantendaten. Durch die Parameterwerte kann die Migrationsdauer im Vergleich zu den Verfahren 1 und 2 weiter reduziert werden. *Abbildung 4b* zeigt jedoch die hohe Mehrbelastung während der Migration aufgrund der langen Sperren bei der Deserialisierung. Der geringe Transaktionsdurchsatz hat zur Folge, dass weniger als 1 % der Seiten durch den Zielserver angefragt werden. Durch die Pufferung von Seiten im Rahmen der Anforderungsphase erreicht das Verfahren bereits wenige Sekunden nach Abschluss der Anforderungsphase die maximale Performance.

Verfahren 4 simuliert das in Kapitel 2 aufgegriffene Verfahren Stop-And-Copy. Nach der Bereitstellung des Schema-Grundgerüsts sorgt die gewählte Paketgröße von 2 GB für die Anforderung sämtlicher Mandantendaten im Rahmen eines einzigen Datenpakets. Dessen Anforderung verhindert eine parallele Transaktionsverarbeitung auf dem Zielserver und verursacht eine 18-sekündige Ausfallzeit. Mangels Übertragung des Pufferstatus hat dieses Verfahren zudem eine Mehrbelastung auf dem Zielserver zur Folge (siehe *Abbildung 4b*). Die maximale Performance für den Mandanten wird in der Folge erst 35 Sekunden nach Abschluss der Anforderungsphase erreicht. Das Verfahren sollte nur bei der Migration inaktiver Mandanten in Betracht gezogen werden.

Die Synchronisationsphase kann bei allen Verfahren in maximal 30 Millisekunden abgeschlossen werden (siehe Abbildung 5). Dies ist insbesondere auf die geringe Transaktionsgröße von 10 Anweisungen zurückzuführen, wodurch der Proxy-Server die Verbindungen schon nach einigen Millisekunden Verarbeitungszeit auf den Zielserver umleiten kann. Entsprechend werden nur wenige Transaktionen parallel ausgeführt, kaum Seiten durch den Zielserver angefordert und keine Seite zurückgefordert. Untersuchungen mit größeren Transaktionen zeigten, dass mehrfaches Übertragen von Seiten bei `DUPLEX_ALL` und `DUPLEX_UNMODIFIED` den Kommunikationsaufwand nur geringfügig erhöht, jedoch zu deutlich weniger Transaktionsabbrüchen führt. Die Ergebnisse weiterer Untersuchungen sind in [Göb15] zu finden.

5 Verwandte Arbeiten

In diesem Kapitel betrachten wir existierende Ansätze zur Live-Migration für mandantenfähige Datenbanksysteme sowie Konzepte in anderen Forschungsgebieten mit vergleichbarem Anforderungsprofil. Das Alleinstellungsmerkmal KitMigs liegt in der Unterstützung verschiedener Migrationsabläufe. Zudem ist es das erste Verfahren für Shared-Nothing-Architekturen, welches auf *Shared Process* basiert. Die existierenden Verfahren umgehen die Herausforderungen gemeinsam genutzter Datenbanken durch das Migrieren kompletter Datenbanken oder Datenbanksysteme.

Elmore et al. [EDAEA11] veröffentlichten mit *Zephyr* ein Verfahren zur Live-Migration in Shared-Nothing-Datenbanksystemen. Der Verlauf des Verfahrens entspricht der Kombination der KitMig-Module `PUSH_WIREFRAME`, `SIMPLEX` und `PULL_ONLY`. Als reaktives Verfahren eignet es sich insbesondere für Serverentlastungen, ist jedoch für die akute Entlastung aufgrund des Verzichts auf die Pufferübertragung nicht empfehlenswert. Da sowohl KitMig als auch Zephyr in das DBMS H2 integriert wurden, haben sie viele Design-Entscheidungen und Konzepte gemein. Zephyr verfolgt das Ziel, Komplexität zu vermeiden, indem es beispielsweise mehrfache Seitenübertragungen strikt verbietet und keine Informationen zu Pufferinhalten an den Zielserver überträgt. Bei diesen Einschränkungen und den resultierenden Schwächen Zephyrs setzt KitMig durch die Bereitstellung diverser Module und Parameter an.

Slacker [BCM+12] ist ein proaktives Verfahren zur Migration kompletter Datenbankinstanzen. Initial erstellt es eine Datenbanksicherung, überträgt sie zum Zielsystem und spielt sie dort ein. Anschließend wird der Zielserver iterativ über Datenänderungen durch verarbeitete Transaktionen informiert. Sind nur noch wenige Änderungen zu übertragen, wird die Transaktionsverarbeitung der Datenbankinstanz gestoppt und ohne Synchronisation auf das Zielsystem umgestellt. Streng genommen führt es somit keine Live-Migration durch. Analog zu KitMig bietet Slacker eine dynamische Drosselung der Seitenübertragungsgeschwindigkeit zur Reduzierung von Mehrbelastungen. Aufgrund seiner proaktiven Vorgehensweise und seines Verzichts auf das Übertragen von Pufferinhalten ist es lediglich für die vorausplanende Serverkonsolidierung geeignet. Als hybrides Verfahren bietet KitMig kein vergleichbares Verfahren.

Das Migrationsverfahren *Albatross* [DNAA10, DNAEA11] wurde für Shared-Disk-Datenbanksysteme entwickelt. Aufgrund des gemeinsamen Speichersystems migriert es keine persistenten Daten, sondern konzentriert sich auf das Verschieben des Pufferinhalts eines Mandanten. Analog zu Slacker überträgt es zu Beginn den kompletten Pufferinhalt und anschließend iterativ dessen Änderungen an den Zielserver. Bei der Lastmigration wird im Gegensatz zu Slacker der Status aktiver Transaktionen migriert, wodurch es sich auch grundsätzlich von allen KitMig-Verfahren unterscheidet, bei denen Transaktionen stets durch einen Server verarbeitet werden. Albatross eignet sich für vorausplanende Serverentlastungen und -konsolidierungen.

ProRea [SCM13] ist das erste veröffentlichte hybride Verfahren zur Live-Migration in Shared-Nothing-Datenbanksystemen und ist vergleichbar mit einem KitMig-Verfahren, bestehend aus `PUSH_BUFFER`, `DUPLEX_ALL` und `PULL_ONLY`. Durch eine zweistufige

Bereitstellungsphase und den Einsatz von Snapshot Isolation verzichtet es auf einer art restriktives Sperrsystem wie KitMig. Dadurch erlaubt es jedoch nicht serialisierbare Ablaufpläne. Beide Verfahren verwalten Eigentumsrechte an Seiten für den Quell- und Zielserver. Die Bereitstellung des kompletten Pufferinhalts eines Mandanten führt gemäß Abschnitt 3.3 bei aktiven Mandanten zu einer späten Lastmigration, weshalb *ProRea* für akute Serverentlastungen ungeeignet und für vorausplanende Entlastungen nur bedingt geeignet ist. Für die Serverkonsolidierung stellt es hingegen einen geeigneten Ansatz dar.

Das Anforderungsprofil der Live-Migration ist in vergleichbarer Form in verschiedenen Forschungsgebieten vorzufinden. Die dabei entwickelten Konzepte und Verfahren können zum Teil auf die Live-Migration in relationalen Datenbanksystemen übertragen werden. Seit geraumer Zeit werden Migrationsverfahren für Betriebssystemprozesse [PM83] untersucht. Unter der Voraussetzung eines gemeinsamen Speichersystems migrieren sie die Arbeitsspeicherinhalte und Prozesszustände an ein Zielsystem. Die Erkenntnisse wurden auf die Live-Migration von VMs übertragen [CFH+05, NLH05]. Für VM-Migrationen zu entfernten Rechnern mittels eines überregionalen Netzwerks (Wide Area Network, WAN) bedarf es der Migration von VMs und persistenter Daten. Bradford et al. [BKFS07] schlugen hierfür ein proaktives Verfahren vor, dessen Ablauf und Drosselungsmöglichkeiten große Ähnlichkeiten zu Slacker aufweisen. Fuzzy Reorganisation [SBC97] zur Reorganisation relationaler Datenbanken im laufenden Betrieb besitzt Herausforderungen, die mit denen der Live-Migration vergleichbar sind. Im laufenden Betrieb wird dabei eine Kopie der Daten erstellt, reorganisiert und unter Berücksichtigung zeitgleich durchgeführter Änderungen durch eine finale Synchronisation in das aktive System integriert.

6 Zusammenfassung und Ausblick

Die Live-Migration von Mandantendaten gehört zu den wesentlichen Herausforderungen mandantenfähiger Datenbanksysteme als Schlüsselkomponente zur Realisierung von Lastverteilung und Elastizität. In diesem Beitrag stellten wir die Migrationslösung KitMig vor, die für die Live-Migration mandantenfähiger Datenbanksysteme mit Shared-Nothing-Architekturen entwickelt wurde. Wir skizzierten die in KitMig verwendeten Phasen und gaben einen Einblick in die Integration des Systems in das Open-Source-DBMS H2. Die Ergebnisse unserer Untersuchungen verdeutlichten, dass eine geeignete Kombination der bereitgestellten Module und Parameterwerte die Anpassung des Migrationsablaufs an verschiedene gegebene Anforderungen ermöglicht.

Der Einsatz von KitMig ist mit Einschränkungen verbunden, deren Aufhebung das Ziel weiterer Arbeiten ist. Die größte Bedeutung kommt dabei der Entwicklung einer logbasierten Systemwiederherstellung zu, um die Dauerhaftigkeit von Transaktionen während der Migration zu gewährleisten. Zudem arbeiten wir an einer iterativen Datenübertragung in der Bereitstellungsphase, um die Dauer der notwendigen Blockade von modifizierenden Transaktionen zu reduzieren. Mögliche Erweiterungen bestehen in der Komprimierung zu übertragender Datenpakete sowie in der Pausierung aktuell nicht ausführbarer Transaktionen unter Bildung einer Warteschlange statt der sofortigen Veranlassung eines Transaktionsabbruchs. Flexible Migrationsverfahren wie KitMig führen zu neuen Herausforderun-

gen für Migrationsstrategien wie der Berücksichtigung der Eigenschaften verschiedener Migrationsabläufe bei der Entscheidungsfindung sowie der Festlegung des geeigneten Migrationsablaufs in Abhängigkeit vom Anforderungsprofil und Systemzustand. Die Herausforderungen zeigen offene Problemstellungen für zukünftige Arbeiten auf.

Literatur

[AEDE11] Divyakant Agrawal, Amr El Abbadi, Sudipto Das und Aaron J. Elmore. Database Scalability, Elasticity, and Autonomy in the Cloud. In *DASFAA*, Seiten 2–15, 2011.

[AJKS09] Stefan Aulbach, Dean Jacobs, Alfons Kemper und Michael Seibold. A Comparison of Flexible Schemas for Software as a Service. In *SIGMOD*, Seiten 881–888, 2009.

[AJPK09] Stefan Aulbach, Dean Jacobs, Jürgen Primsch und Alfons Kemper. Anforderungen an Datenbanksysteme für Multi-Tenancy- und Software-as-a-Service-Applikationen. In *BTW*, Seiten 544–555, 2009.

[BCM+12] Sean Barker, Yun Chi, Hyun Jin Moon, Hakan Hacıgümüş und Prashant Shenoy. „Cut Me Some Slack“: Latency-aware Live Migration for Databases. In *EDBT*, Seiten 432–443, 2012.

[BKFS07] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann und Harald Schiöberg. Live Wide-area Migration of Virtual Machines Including Local Persistent State. In *VEE*, Seiten 169–179, 2007.

[BZP+10] Cor-Paul Bezemer, Andy Zaidman, Bart Platzbeecker, Toine Hurkmans und Aad 't Hart. Enabling Multi-Tenancy: An Industrial Experience Report. In *ICSM*, Seiten 1–8, 2010.

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt und Andrew Warfield. Live Migration of Virtual Machines. In *NSDI*, Seiten 273–286, 2005.

[CJP+11] Carlo Curino, Evan Jones, Raluca Popa, Nirmesh Malviya, Eugene Wu, Samuel Madden, Hari Balakrishnan und Nickolai Zeldovich. Relational Cloud: A Database Service for the Cloud. In *CIDR*, Seiten 235–240, 2011.

[CJZM10] Carlo Curino, Evan Jones, Yang Zhang und Sam Madden. Schism: a Workload-Driven Approach to Database Replication and Partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57, September 2010.

[CST+10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan und Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *SoCC*, Seiten 143–154, 2010.

[DNAA10] Sudipto Das, Shoji Nishimura, Divyakant Agrawal und Amr El Abbadi. Live Database Migration for Elasticity in a Multitenant Database for Cloud Platforms. Bericht 2010-09, University of California, Santa Barbara, 2010.

[DNAEA11] Sudipto Das, Shoji Nishimura, Divyakant Agrawal und Amr El Abbadi. Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud Using Live Data Migration. *Proc. VLDB Endow.*, 4(8):494–505, 2011.

[EDAA11] Aaron J. Elmore, Sudipto Das, Divyakant Agrawal und Amr El Abbadi. Towards an Elastic and Autonomic Multitenant Database. In *NetDB (SIGMOD-Workshop)*, 2011.

[EDAEA11] Aaron J. Elmore, Sudipto Das, Divyakant Agrawal und Amr El Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In *SIGMOD*, Seiten 301–312, 2011.

[EDP+13] Aaron J. Elmore, Sudipto Das, Alexander Pucher, Divyakant Agrawal, Amr El Abbadi und Xifeng Yan. Characterizing Tenant Behavior for Placement and Crisis Mitigation in Multitenant DBMSs. In *SIGMOD*, Seiten 517–528, 2013.

[Göb14a] Andreas Göbel. H2 Proxy – Dynamic Load Balancing for Multi-Tenant Database Systems. *it – Information Technology*, 56(3):127–133, 2014.

[Göb14b] Andreas Göbel. MuTeBench: Turning OLTP-Bench into a Multi-Tenancy Database Benchmark Framework. In *CLOUD COMPUTING*, Seiten 84–87, 2014.

[Göb15] Andreas Göbel. Flexible Live-Migration zur dienstgütebasierten Ressourcenverwaltung in mandantenfähigen Datenbanksystemen. Dissertation (in Vorbereitung), Friedrich-Schiller-Universität Jena, 2015.

[JA07] Dean Jacobs und Stefan Aulbach. Ruminations on Multi-Tenant Databases. In *BTW*, Seiten 514–521, 2007.

[LHM+13] Ziyang Liu, Hakan Hacıgümüş, Hyun Jin Moon, Yun Chi und Wang-Pin Hsiung. PMAX: Tenant Placement in Multitenant Databases for Profit Maximization. In *EDBT*, Seiten 442–453, 2013.

[NLH05] Michael Nelson, Beng-Hong Lim und Greg Hutchins. Fast Transparent Migration for Virtual Machines. In *ATEC*, Seiten 391–394, 2005.

[PM83] Michael L. Powell und Barton P. Miller. Process Migration in DEMOS/MP. *SIGOPS Operating Systems Review*, 17(5):110–119, 1983.

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

[Rei10] Berthold Reinwald. Database Support for Multi-tenant Applications. *WISS (ICDE Workshop)*, 1:2, 2010.

[SBC97] G. H. Sockut, T. A. Beavin und C.-C. Chang. A Method for On-line Reorganization of a Database. *IBM Systems Journal*, 36(3):411–436, 1997.

[SCM13] Oliver Schiller, Nazario Cipriani und Bernhard Mitschang. ProRea: Live Database Migration for Multi-tenant RDBMS with Snapshot Isolation. In *EDBT*, Seiten 53–64, 2013.

[SJK+13] Jan Schaffner, Tim Januschowski, Megan Kercher, Tim Kraska, Hasso Plattner, Michael J. Franklin und Dean Jacobs. RTP: Robust Tenant Placement for Elastic In-memory Database Clusters. In *SIGMOD*, Seiten 773–784, 2013.

[SK12] Michael Seibold und Alfons Kemper. Database as a Service. *Datenbank-Spektrum*, 12(1):59–62, 2012.

[WB09] Craig D. Weissman und Steve Bobrowski. The Design of the Force.Com Multitenant Internet Application Development Platform. In *SIGMOD*, Seiten 889–896, 2009.

[YQR+12] Tao Yu, Jie Qiu, Berthold Reinwald, Lei Zhi, Qirong Wang und Ning Wang. Intelligent Database Placement in Cloud Environment. In *ICWS*, Seiten 544–551, 2012.

[YSY09] Fan Yang, Jayavel Shanmugasundaram und Ramana Yerneni. A Scalable Data Platform for a Large Number of Small Applications. In *CIDR*, 2009.