SLO-basiertes Management in relationalen Datenbanksystemen mit nativer Multi-Tenancy-Unterstützung

Andreas Göbel
Lehrstuhl für Datenbanken und Informationssysteme
Fakultät für Mathematik und Informatik
Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, 07743 Jena
andreas.goebel@uni-jena.de

KURZFASSUNG

Das an Bedeutung und Akzeptanz gewinnende Geschäftsmodell Software as a Service ermöglicht Unternehmen die Konzentration auf ihr Kerngeschäft durch das Beziehen von Dienstleistungen über das Internet. Die Festlegung von Service Level Agreements gewährleistet eine hohe Qualität des Dienstes. Um die operationalen Kosten des Service Providers gering zu halten, bedarf es des Einsatzes einer Multi-Tenancy-Architektur, die zu neuen Herausforderungen für den Einsatz von Datenbankverwaltungssystemen führt.

In diesem Beitrag werden die Problemstellungen der Realisierung von Multi Tenancy in heutigen Datenbankverwaltungssystemen aufgezeigt und eine native Unterstützung von Multi Tenancy in jenen Systemen motiviert. Es wird hervorgehoben, dass die Integration von mandantenspezifischen Service Level Agreements zur Steigerung der Qualität des Dienstes beiträgt. Hierzu wird die Verwendung dieser bereitgestellten Daten zur Ressourcenverwaltung und -überwachung sowie der Lastverteilung von Mandanten verdeutlicht.

Kategorien und Themenbeschreibung

H.2.7 [**Database Management**]: Database Administration—*Data dictionary/directory*; H.2.1 [**Database Management**]: Logical Design—*Schema and subschema*

Allgemeine Begriffe

Design, Management, Measurement

Stichworte

Relational Database, Multi Tenancy, Service Level Agreement

24^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 29.05.2012 - 01.06.2012, Lübbenau, Germany.

1. EINLEITUNG

Software as a Service (SaaS) bezeichnet ein Geschäftsmodell, bei dem Unternehmen von einem Drittanbieter Anwendungen in Form von Services über das Internet beziehen. Der Ansatz steht dem herkömmlichen On-Premise-Modell gegenüber, bei dem Unternehmen die Lizenzen für Software käuflich erwerben und die Anwendungen anschließend auf eigener Hardware betreiben. Durch das Betreiben und Warten der Anwendung sowie der notwendigen Hardware durch den Drittanbieter (SaaS-Anbieter) können die als Mandanten bezeichneten Unternehmen auf den Erwerb und die Wartung der Hardware verzichten. Service Level Agreements (SLAs) legen dabei als Bestandteil der Dienstleistungsverträge durch die Definition verschiedener Service Level Objects (SLOs) die Qualität der Services fest.

Laut Studien renommierter Marktanalyseunternehmen wird die Bedeutung von SaaS in den nächsten Jahren weiter zunehmen. So prognostiziert Pierre Audoin Consultants SaaS für das Jahr 2015 einen Umsatzanteil am Software-Markt in Höhe von zehn Prozent und begründet die zunehmende Bedeutung u.a. mit entsprechenden Anpassungen der Produktportfolios sowie Akquisitionen von bedeutenden Software-Anbietern wie Oracle, SAP und Microsoft [18].

SaaS-Angebote sind aktuell insbesondere in den Bereichen E-Commerce, Kundenbeziehungsmanagement und bei kollaborativen Aufgaben wie E-Mail zu finden. Zunehmend werden Produkte in den Bereichen Humankapital-Management und Unternehmensressourcenplanung angeboten. Um einen möglichst großen Markt ansprechen zu können, ermöglichen SaaS-Anbieter eine individuelle Anpassung des Services. Die Angebote richten sich auch an kleine und mittlere Unternehmen (KMU), denen aufgrund begrenzter finanzieller Möglichkeiten die Mittel für den Erwerb und Support einer vergleichbaren On-Premise-Lösung fehlen. Dieser so genannte Long Tail [3] kann durch Preisvorteile gegenüber On-Premise-Angeboten und entfallende Kosten der Mandanten für die Beschaffung und Wartung von Hardware bedient werden [10]. Es bedarf hierzu einer hohen Wirtschaftlichkeit des Services durch eine Konsolidierung von Mandanten auf physischen Ressourcen in Form von Multi-Tenancy-Anwendungen.

2. MULTI TENANCY

Die Architektur von SaaS-Angeboten ist in großem Maße vom Geschäftsmodell des Anbieters abhängig. Sie wird beispielsweise beeinflusst durch das zur Verfügung stehende

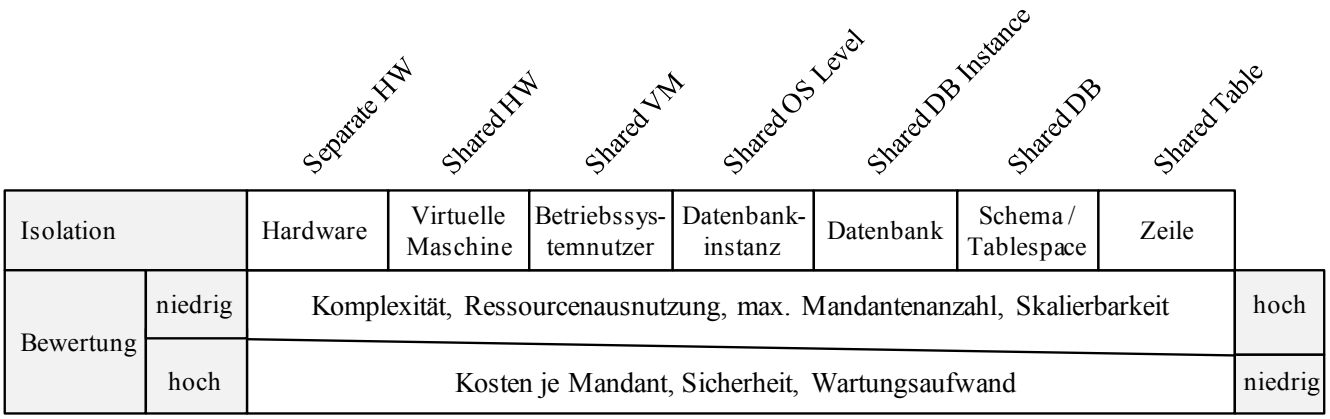

Abbildung 1: Ansätze zur Mandantenisolation im DB-Layer, angelehnt an [19]

Entwicklungskapital, den Zielmarkt sowie die Anzahl und Charakteristika der Mandanten. Zu den Charakteristika gehören u.a. die benötigte Datenmenge, die voraussichtliche Workload und die Anforderungen der Mandanten bezüglich Verfügbarkeit, Performance, Datenisolation und Individualisierung der Anwendung.

Um die in der Einleitung motivierte hohe Wirtschaftlichkeit eines SaaS-Angebots zu erzielen, werden vermehrt Multi-Tenancy-Architekturen eingesetzt. Diese erlauben allen Service-Nutzern die gemeinsame Verwendung von Hardware-Ressourcen durch das Anbieten **einer** gemeinsamen Anwendungsinstanz [9]. Sowohl auf der Applikations- als auch der Datenbankseite sind verschiedene Ansätze zur Separierung von Mandanten denkbar.

2.1 Multi Tenancy in DBMS

Es existiert ein breites Realisierungspektrum für Multi Tenancy in einem Datenbankserver. In Abbildung 1 werden die wichtigsten Ansätze zusammengefasst sowie Vorteile und Herausforderungen aufgezeigt. Neben der Möglichkeit, auf eine Mandantenkonsolidierung zu verzichten (*Separate Hardware*), können Mandanten beispielsweise durch die Zuweisung separater virtueller Maschinen (*Shared HW*), durch die Nutzerverwaltung des Betriebssystems (*Shared VM*) oder durch die Verwendung getrennter Datenbankinstanzen (*Shared OS Level*) bzw. Datenbanken (*Shared DB Instance*) voneinander isoliert und dennoch auf einem Datenbankserver verwaltet werden. Aufgrund des initialen Ressourcenbedarfs und der gesonderten Administration dedizierter Datenbanken, Datenbankinstanzen, Betriebssysteme oder virtueller Maschinen führen diese Ansätze für den SasS-Anbieter zu hohen Kosten je Mandant. Sie sollten daher nur bei zwingender Notwendigkeit eines hohen Isolations- bzw. Sicherheitslevels oder bei einer geringen Anzahl von Mandanten verwendet werden.

Der Ansatz *Shared Database* erlaubt durch die Mandantenkonsolidierung in einer Datenbank die gemeinsame Nutzung von Datenbankprozessen und Hauptspeicherinhalten. Die Zuweisung zu eigenen Datenbankobjekten wie Tabellen und Indizes führt bei Nutzung der Datenbankzugriffskontrollen zu einer logischen Isolation der Mandanten. Durch die Zuweisung der mandantenspezifischen Objekte zu separaten Speicherorten (Tablespaces) kann zudem eine physische Trennung erreicht werden. Das Hinzufügen und Löschen von Mandanten sowie mandantenspezifische Schemaänderungen bedürfen bei diesem Ansatz das Absetzen von DDL-Statements, die bei einigen DBMS zu Problemen mit dem fortlaufenden Betrieb führen können. Die hohe Anzahl an Tabellen führt zudem zu einem hohen Hauptspeicherbedarf sowie partiell gefüllten Seiten des Datenbankpuffers. [6, 12]

Bei dem Ansatz *Shared Table* werden Objekte der Datenbank von Mandanten gemeinsam genutzt. Tabellen enthalten somit Tupel verschiedener Mandanten, weshalb die Verwendung einer zeilenbasierten Zugriffskontrolle nötig ist. Eine zusätzliche Tabellenspalte legt hierbei die Zugehörigkeit des Tupels zum entsprechenden Mandanten fest. Durch den Verzicht auf mandantenspezifische Datenbankobjekte ist die Größe des Datenbankkatalogs nahezu unabhängig von der Mandantenanzahl. Die maximale Anzahl unterstützter Mandanten ist somit laut [12] lediglich durch die maximale Anzahl unterstützter Tabellenzeilen beschränkt, was den Ansatz für das Bedienen des im Abschnitt 1 angesprochenen Long Tails prädestiniert. Durch die hohe Ressourcenausnutzung reduziert sich der Overhead bezüglich Fest- und Hauptspeicherbedarf pro Mandant auf ein Minimum. Administrative Operationen und Updates der Anwendung werden bei diesem Ansatz in der Regel für alle Mandanten ausgeführt, was den Wartungsaufwand des Anbieters reduziert, die Individualität des Services jedoch einschränkt. So können die in [11] angesprochenen individuellen Anforderungen von Mandanten in Bezug auf Aspekte der Datenbankadministration und -konfiguration wie Backup-Strategien und Archivierungsintervalle, die Replikationsart und Replikatanzahl oder Vorgaben zur Arbeitsweise des Datenbankoptimierers nicht erfüllt werden. Aufgrund der Konsolidierung von vielen Mandantendaten innerhalb einer Tabelle liegen die größten Herausforderungen dieses Ansatzes in der Gewährleistung der Isolation der Mandantendaten und -Performance sowie der Erarbeitung eines Datenbankschemas, welches den Mandanten die Anpassung der vom SaaS-Anbieter zur Verfügung gestellten Anwendungen erlaubt.

2.2 Schemaflexibilität

Um seinen Service einer möglichst breiten Zielgruppe anbieten zu können, sind SaaS-Anbieter bemüht, Mandanten weitreichende Anpassungsmöglichkeiten zu bieten. Diese umfassen laut [10] unter anderem eine Anpassung der Benutzer-

oberfläche im Sinne eines Corporate Identity, die Anpassung an Geschäftsabläufe durch Modifikation von Geschäftsregeln, individuelle Regelungen bezüglich der Zugangskontrollen sowie die Möglichkeit zur Erweiterung des Datenbankschemas durch zusätzliche Tabellenspalten oder komplette Tabellen. Diese Anforderung stellt sich aufgrund des notwendigen Zusammenführens individueller Datenbankschemata der Mandanten auf ein Gesamtschema der Datenbank insbesondere beim Ansatz *Shared Table* als Herausforderung dar. Aulbach et al. stellen in [4, 5] eine Reihe von Ansätzen vor, die in folgende Kategorien eingeteilt werden können:

Vertikale Speicherung: Dieser Ansatz basiert auf dem Entity-Attribute-Value-Modell [17], welches beispielsweise im medizinischen Bereich Anwendung findet. Jeder Attributwert eines mandantenspezifischen Datensatzes wird auf einen Datensatz des Gesamtschemas abgebildet, der zur Identifizierung neben dem Attributwert den zugehörigen Mandanten-, Tabellen- und Spaltenname sowie die Zeilennummer enthält. Ein mandantenspezifischer Datensatz muss hierbei zur Laufzeit durch Verbundoperationen erzeugt werden, um die Attributwerte zu einem Datensatz zusammenzufügen.

Horizontale Speicherung: Ein mandantenspezifischer Datensatz wird direkt auf Datensätze des Gesamtschemas abgebildet. Flexibilität kann beispielsweise durch die Nutzung einer universellen Tabelle geboten werden, welche durch eine Vielzahl von generischen Spalten mit flexiblen Datentypen die Speicherung beliebiger Datensätze erlaubt und die Verwaltung des Schemas in die Anwendung verschiebt.

XML: Heutige Datenbankmanagementsysteme bieten zunehmend native Unterstützung des XML-Datentyps zur Speicherung semistrukturierter Daten. Durch dessen Verwendung können verschiedene mandantenspezifische Schemata innerhalb einer Tabelle verwaltet werden.

Hybride Speicherung: Die vorherigen Speicherformen können miteinander verbunden werden, um ihre Stärken zu kombinieren.

2.3 Native Unterstützung durch DBMS

Die in Abschnitt 2.2 vorgestellten Ansätze zur Unterstützung individueller Mandantenschemata können bei heutigen Datenbanksystemen nur mit Hilfe einer überhalb des DBMS liegenden Schicht zur Transformation der mandantenspezifischen Datenbankanfragen und vom DBMS erhaltenen Ergebnisse realisiert werden. Diese Schicht regelt die Zugriffskontrolle und verwaltet die Schemata der Mandanten, sodass die Schemainformationen vom DBMS nicht zur Optimierung genutzt werden können. Zudem führt sie zu erhöhten Wartungsaufwand und beeinflusst unter Umständen die Skalierbarkeit des Systems. [6, 20]

Bisherige Konzepte und prototypische Implementierungen [6, 20] zur nativen Unterstützung von Mandanten in relationalen Datenbanksystemen verlagern im Wesentlichen die Transformationsschicht inklusive der benötigten Metadaten ins Datenbanksystem. Sie verfolgen hiermit das Ziel einer effizienten Mandantenkonsolidierung sowie der Abbildung mandantenspezifischer Schemata auf ein Gesamtschema, um die oben aufgezeigten Nachteile einer externen Transformation anzugehen. Des Weiteren wird in [7] ein Konzept vorgestellt, welches Mandanten durch die Unterstützung mehrerer paralleler Basisschema-Versionen einen verzögerten Wechsel auf eine neue Version der SaaS-Applikation und dessen Erweiterungen erlaubt.

3. SLO-BASIERTE VERWALTUNG

Ein bisher kaum betrachtetes, jedoch nicht minder bedeutsames Forschungsgebiet im Zusammenhang mit der nativen Unterstützung von Multi Tenancy in Datenbanksystemen ist die Integration von abgeleiteten Richtlinien aus den Service Level Agreements (SLAs) der Mandanten sowie die Verwaltung der Mandantendaten auf Basis jener Richtlinien. Durch die zentrale Haltung der Richtlinien als Bestandteil des Datenbankkatalogs können sie von verschiedenen DBMS-Komponenten und externen Werkzeugen zur Verbesserung der Dienstqualität verwendet werden.

SLO-basiertes Management kann mittels automatisiertem und proaktivem Agieren den Administrationsaufwand für den SaaS-Anbieter reduzieren. Zudem kann es die Lastverteilung und Migration von Mandanten unterstützen, um Mandanten stets die Ressourcen zur Verfügung zu stellen, die ihren Anforderungen genügen. Die Priorisierung von Mandanten bei der Verarbeitung ihrer Systemanfragen kann durch eine SLO-basierte Ressourcenverwaltung und -überwachung realisiert werden. Das SLO-basierte Management ist somit ein essentielles Mittel, um auf der einen Seite die Betriebskosten der SaaS-Anbieter durch eine hohe Ressourcenausnutzung gering zu halten und auf der anderen Seite den Mandanten eine möglichst hohe Service-Qualität zu bieten.

Die Einsetzbarkeit in verschiedenen Aufgabenfeldern sowie die damit verbundenen Vorzüge für Administratoren und Mandanten begründen intensive Forschung in folgenden Themengebieten:

- Ableitung von geeigneten mandantenspezifischen Richtlinien aus Dienstleistungsverträgen,
- Repräsentation der Richtlinien im Katalog des Datenbankverwaltungssystems,
- Omnipräsentes Monitoring der Einhaltung von Richtlinien,
- Entwicklung und Realisierung geeigneter Algorithmen zur Sicherstellung der Richtlinien.

3.1 Service Level Objects

Als Bestandteil des Dienstleistungsvertrags zwischen dem SaaS-Anbieter und den Mandanten legen Service Level Agreements die zugesicherte Qualität des SaaS-Angebots fest. Hierzu spezifizieren sie beispielsweise Kennzahlen oder Abstufungen bezüglich der folgenden Anforderungen an den Services in Form von Richtlinien, welche als Service Level Objects (SLOs) bezeichnet werden.

- Verfügbarkeit: Systemzugänglichkeit, Wartungsfenster, Wiederherstellungszeiten in Fehlerfällen
- Performance: Geschwindigkeit der Datenverarbeitung, Reaktionszeiten der Schnittstellen
- Sicherheit: Datenschutz, Datensicherheit, Art der Isolation von anderen Mandanten

Die SLOs sollten u.a. aussagekräftig, erreichbar, messbar und verständlich sein [22]. Bestandteil der SLAs sind neben den

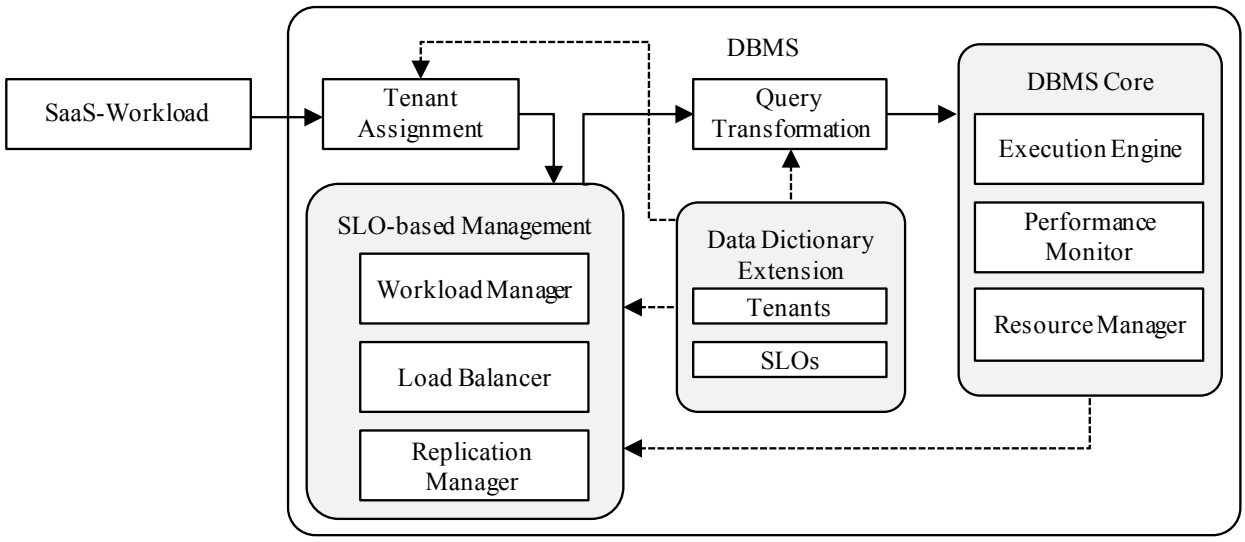

Abbildung 2: Integration von SLO-basiertem Management ins DBMS

SLOs entsprechende Regelungen für den Fall, dass der SaaS-Anbieter die zugesicherten SLOs nicht erfüllen kann. In der Regel erfolgt dies über gestaffelte Strafzahlungen oder lediglich über die Verminderung der anfallenden Grundgebühren für Mandanten.

3.2 SLO-Repräsentation

Nach der Einigung der SaaS-Anbieter und Mandanten über zu gewährleistende SLOs und dem Vertragsabschluss erfolgt die Überführung der finalen Bestimmungen in Anforderungen an die zugrunde liegende Technologie und Hardware des Anbieters. SLOs werden vorrangig technologieunabhängig definiert und gelten in der Folge für den gesamten Service, weshalb eine adäquate Ableitung von mandantenspezifischen Richtlinien für die einzelnen Systemkomponenten wie dem Datenbanksystem eine bedeutsame und komplexe Aufgabe darstellt. Zudem verdeutlicht dieser Sachverhalt, dass die Einhaltung jener Richtlinien, analog zu Multi Tenancy, in allen Schichten des Gesamtsystems von Bedeutung ist. Mit zunehmender Mandantenkonsolidierung auf den zur Verfügung stehenden Systemressourcen nimmt die Beeinflussung unter Mandanten bezüglich der Performance zu, was die Erstellung passender Richtlinien weiter erschwert.

Die resultierenden Richtlinien sollten entsprechend einem adäquaten Modell erstellt werden. Sie bestehen aus einer Kombination aus Anforderungen, beispielsweise bezüglich der Performance, Verfügbarkeit oder Sicherheit sowie einer Priorität. Diese spiegelt die Bedeutsamkeit des Mandanten für das Unternehmen wider und basiert typischerweise auf der Größenordnung der entsprechenden Vertragsstrafen eines Mandanten [14]. Unter Umständen können hierbei jedoch Aspekte eine Rolle spielen, die nicht direkt aus dem Dienstleistungsvertrag abgeleitet werden können wie die Reputation des Mandanten oder seine Bedeutung als strategischer Partner des SaaS-Anbieters.

Die native Unterstützung von Multi Tenancy im DBMS bringt in der Regel eine Katalogerweiterung [20] um Tabellen zur Repräsentation der Mandanten und der zugehörigen Datenbankobjekte wie Tabellen oder Indizes mit sich. Die Definition und anschließende Überwachung der SLOs bedarf wiederum einer Erweiterung des Katalogs um die SLOs jedes Mandanten sowie seiner zugewiesenen Bedeutsamkeit bzw. Priorität. Durch die Zuordnung eines Mandanten zu einer SLO-Kategorie wie 'Standard', 'Premium' oder 'Individuell' kann bei der SLO-Zuweisung der Overhead bezüglich des Fest- und Hauptspeicherbedarfs je Mandant reduziert werden. Abbildung 2 verdeutlicht diese Erweiterung und kennzeichnet, dass Katalogerweiterungen beispielsweise für die Zuweisung von Mandanten zu Anfragen (*Tenant Assignment*) und die in Abschnitt 2.3 angesprochene Transformationskomponente (*Query Transformation*) benötigt wird.

3.3 Workload Management

Einige Datenbankverwaltungssysteme bieten mittels Workload Management die Möglichkeit zur Überwachung von Abfragen und den von ihnen benötigten Ressourcen. IBM DB2 for Linux, UNIX and Windows stellt hierfür beispielsweise den DB2 Workload Manager [1] und Oracle den Oracle Database Resource Manager und Oracle Scheduler [2] bereit. Diese Anwendungen bieten ein breites Spektrum an Mitteln zur Überwachung von Anfragen, welches u.a. das Aufteilen von Systemressourcen zwischen Abfragen, die Priorisierung, Ab- und Unterbrechung von Abfragen sowie eine Zeitplanung von Abfragen enthalten kann.

Das DBMS muss fortwährend den Auslastungszustand der Ressourcen messen und für das Workload Management bereitstellen. Abbildung 2 verdeutlicht, mit welchen Komponenten des DBMS-Kerns der Workload Manager typischerweise kommuniziert, um die Verarbeitung von Abfragen zu steuern und zu überwachen [14]:

- *Execution Engine* (Verwaltung der Ausführung von Abfragen),
- *Performance Monitor* (Überwachung der Verarbeitung von Abfragen),
- *Resource Monitor* (Regelung der Ressourcenallokation der Abfragen).

Die Erweiterung des Datenbankkatalogs um Mandanten und SLOs stellt dem Datenbankverwaltungssystem die nöti-

gen Mittel zur Verfügung, um neben der korrekten Ausführung nebenläufiger Transaktionen eine auf SLOs basierende Parallelisierung von Mandanten zu erzielen. Verschiedene Nutzungsprofile und -zeiträume der Mandanten sowie übliche Nutzungsschwankungen erlauben eine Überbuchung der zur Verfügung stehenden physischen Systemressourcen wie der CPU, dem Hauptspeicher oder der Bandbreite zum Festspeicher. Dies ermöglicht eine ausgezeichnete Auslastung der Ressourcen und hält somit die operativen Kosten des SaaS-Anbieters gering. Im (Ausnahme-)Fall hoher Lastspitzen durch einen parallelen intensiven Zugriff einer Großzahl von Mandanten, welche die Ressourcen gemeinsam nutzen, kann die Einhaltung aller offerierten SLOs nicht gewährleistet werden. Lastspitzen können aufgrund von regelmäßigen Vorgängen wie Gehaltsbuchungen oder beispielsweise aufgrund von unvorhergesehenen Nutzungsschwankungen unregelmäßig auftreten [6]. Um die Häufigkeit dieser Situation zu reduzieren und beim Auftreten adäquat zu reagieren, bedarf es geeigneter Strategien zur Ablaufplanung der Abfragen sowie der Ressourcenzuteilung. Aus ökonomischer Sicht des Anbieters gilt es hierbei, die Summe der zu zahlenden Vertragsstrafen zu minimieren. In [13] wird dieses Ziel durch einen dynamischen Controller und der Zuhilfenahme zweier Kostenfunktionen verfolgt, womit zudem eine dauerhafte Übererfüllung der SLOs von Mandanten mit hoher Priorität auf Kosten von Mandanten mit geringer Priorität vermieden wird.

Häufig wird in diesen Modellen lediglich die Minimierung von Strafzahlen bezüglich der aktuellen Überauslastung betrachtet und die Korrelation von Entscheidungen bei verschiedenen Lastspitzen außer Acht gelassen. So ist es möglich, dass die SLOs eines Mandant mit geringer Priorisierung bei Lastspitzen wiederholt nicht erreicht werden können. Aus ökonomischer Sicht ist dies für den SaaS-Anbieter augenscheinlich eine optimale Strategie, sie kann jedoch zur Verärgerung oder gar Kündigung des Mandanten führen. Folglich gilt es, frühere Entscheidungen in die Priorisierung von Mandanten und Ressourcenzuweisung bei Lastspitzen einzubeziehen. Dieses Ziel kann nur mit Hilfe einer dynamischen Priorisierung erreicht werden.

Neben der Ablaufsteuerung und Überwachung von Abfragen sowie dem Eingreifen im Falle einer Überlastung besteht eine wesentliche Aufgabe des SLO-basierten Managements in dem Verhindern von häufigen Überlastungen eines Systems. Hierzu sind die Ressource sowie der Schweregrad, die Häufigkeit und eine gegebenenfalls existierende Regelmäßigkeit der Überlastungen zu beobachten. Durch ein proaktives Vorgehen kann zukünftigen Überlastungen vorgebeugt werden, indem Maßnahmen autonom ergriffen werden oder der Administrator in Form von Hinweisen bei seiner Tätigkeit unterstützt wird. Ein mögliches Vorgehen ist die Zuweisung von Mandanten zu anderen physischen Ressourcen durch das Verschieben auf einen anderen Server mit dem Ziel einer Lastverteilung.

3.4 Verteilung von Mandantendaten

Ein SaaS-Anwendung sollte die maximale Anzahl an unterstützten Mandanten möglichst nicht beschränken. Um auch bei vielen Mandanten eine ausreichende Performance zu erreichen, müssen die Daten der Mandanten mittels Tabellen- und Indexpartitionierung auf verschiedene Festspeicher oder gemäß eines verteilten Datenbanksystems auf verschiedene Datenbankknoten verteilt werden. Die Verteilung der Mandanten durch einen *Load Balancer* kann zudem gemäß Abschnitt 3.3 zur Abfederung von Lastspitzen genutzt werden. Entgegen statischer Ansätze [15] zur Berechnung einer optimalen Verteilung von Mandantendaten auf eine Menge von Datenbankknoten sollte die Lastverteilung analog zum Workflow Management dynamisch agieren.

Um unnötige Kommunikation zwischen Datenbankknoten zu vermeiden, sind die Daten eines Mandanten nicht über mehrere Knoten zu verteilen. Dies sollte nur möglich sein, wenn die Anforderungen des Mandanten die zur Verfügung stehenden Ressourcen des Rechners übersteigen.

Die Mandanten legen durch die Nutzung einer SaaS-Anwendung ihre Daten in die Hände des Anbieters und dessen auf Sicherheit spezialisierte IT-Abteilung [11]. Durch die Verteilung von Mandanten auf verschiedene Rechner ist neben dem verringerten Einfluss der Mandanten bezüglich ihrer Performance (Performance-Isolation zwischen Mandanten [8]) ein höherer Isolationsgrad der auf dem Festspeicher abgelegten Mandantendaten erreichbar bis hin zu einer physischen Isolation. Der Isolationsgrad spiegelt sich zum Teil in entsprechenden SLOs der Dienstverträge wider. Er kann durch die in Abbildung 2 dargestellte SLO-Katalogerweiterung im DBMS hinterlegt und vom Lastbalancierer bei der Verteilung der Mandanten auf Rechner berücksichtigt werden.

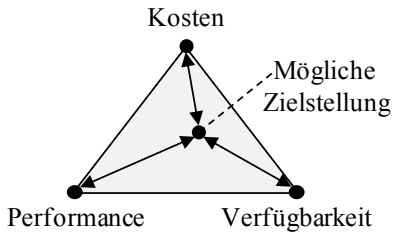

Abbildung 3: Zielkonflikte des SaaS-Anbieters

Verfügbarkeit besitzt gemäß Abschnitt 3.1 eine außerordentliche Bedeutsamkeit bei Dienstleistungsverträgen im SaaS-Umfeld. Ist der Dienst für die Mandanten nicht erreichbar, so kann dies schwerwiegende Folgen haben, die von einer Verzögerung der Arbeitsprozesse bei Mandanten bis hin zu finanziellen Einbußen reichen. Entsprechend werden in den Dienstleistungsverträgen nur geringe (geplante und ungeplante) Ausfallzeiten des Services zugelassen, bei deren Verletzung der SaaS-Anbieter mit erheblichen Strafzahlungen rechnen muss. Entsprechend liegt es im Interesse des Anbieters, eine hohe Verfügbarkeit des Dienstes sicherzustellen. Abbildung 3 verdeutlicht, dass die Erreichung eines hohen Verfügbarkeitsniveaus auf der einen Seite zu zusätzlichen Kosten für den Anbieter führt und auf der anderen Seite die Performance des Services einschränkt. Eine hohe Verfügbarkeit fordert das Vermeiden von Single Points of Failures und kann beispielsweise durch verschiedene Formen der Replikation erreicht werden. Der SaaS-Anbieter könnte durch einen *Replication Manager* die Art und den Umfang der Replikation von Mandantendaten aufgrund von verschiedenen SLOs der Mandanten variieren, um individuelle Anforderungen zu unterstützen.

3.5 SLA-Management in DaaS

Die Verwendung von Datenbanksystemen zur Datenverwaltung einer SaaS-Anwendung stellt nur eine Möglichkeit zur Bereitstellung von Datenbanksystemen als Dienst inner-

halb einer Cloud dar, was als Database as a Service (kurz DaaS oder DbaaS [21]) betitelt wird. Sowohl in kommerziellen DaaS-Angeboten als auch in der DaaS-Forschung spielt Multi Tenancy eine bedeutende Rolle, um die operationalen Kosten von DaaS-Anbietern zu senken. Die Bereitstellung von Daten für eine SaaS-Anwendung unterscheidet sich jedoch im Zusammenhang mit Multi Tenancy erheblich von anderen Dienstmodellen.

- Mandanten in einer SaaS-Anwendung besitzen und erweitern ein gemeinsames Basisschema und greifen zudem in der Regel lesend auf gemeinsame Anwendungsdaten zu. Bei anderen DaaS-Dienstmodellen existieren meist keine mandantenübergreifenden Daten und keine oder vernachlässigbare Ähnlichkeit der Datenbankschemata von Mandanten. Dies wirkt sich auf die Konsolidierungsmöglichkeiten innerhalb einer Datenbank aus.

- SaaS-Anwendungen bestimmen die Art der Workload für das verwendete Datenbanksystem, was sich das SLO-basiertes Management zu Nutze machen kann. Bei anderen DaaS-Dienstmodellen ist die Workload hingegen mandantenspezifisch und unvorhersehbar.

- SLOs sind bei SaaS-Angeboten mit der kompletten Anwendung verknüpft, während bei anderen DaaS-Dienstmodellen meist konkrete Vorgaben für das DBMS definiert sind.

Diese Punkte verdeutlichen, dass Ergebnisse aktueller Forschung im Bereich SLO-basierter Hardware-Provisionierung und Steuerung der Abfrageverarbeitung für DaaS [16] nur sehr eingeschränkt auf Datenbankdienste für SaaS-Anwendungen übertragen werden können und gesonderte Forschung für jenen Bereich vonnöten ist.

4. ZUSAMMENFASSUNG UND NÄCHSTE SCHRITTE

In diesem Beitrag wurde gezeigt, dass eine Integration von Service Level Objects in ein Multi-Tenancy-Datenbanksystem in Form einer Erweiterung des Datenbankkatalogs von verschiedenen DBMS-Komponenten verwendet werden kann, um die Überwachung der SLOs während des Betriebs zu gewährleisten. Als mögliche Verwendungsbeispiele wurden das Workload Management, die Lastverteilung und eine individuelle Replikationssteuerung aufgeführt.

In den weiteren Arbeiten soll der Entwurf der SLO-Integration konkretisiert, verschiedene Implementierungsvarianten verglichen und die Realisierbarkeit anhand eines Prototyps gezeigt werden. Anschließende Performance-Tests sollen Aufschluss darüber geben, ob der zusätzliche Overhead durch die mandantenspezifischen Metadaten und das SLO-basierte Management die Skalierbarkeit und den laufenden Betrieb des Datenbanksystems beeinträchtigen.

5. LITERATUR

[1] *DB2 Workload Manager for Linux, Unix, and Windows*. IBM Corp., 2008.

[2] *Oracle® Database Administrator's Guide 11g Release 2*. Oracle, 2011.

[3] C. Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[4] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD*, pages 1195–1206. ACM, 2008.

[5] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold. A comparison of flexible schemas for software as a service. In *SIGMOD*, pages 881–888. ACM, 2009.

[6] S. Aulbach, D. Jacobs, J. Primsch, and A. Kemper. Anforderungen an Datenbanksysteme für Multi-Tenancy- und Software-as-a-Service-Applikationen. In *BTW*, pages 544–555. GI, 2009.

[7] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper. Extensibility and Data Sharing in evolving multi-tenant databases. In *ICDE*, pages 99–110, 2011.

[8] D. Banks, J. Erickson, M. Rhodes, and J. S. Erickson. Multi-tenancy in Cloud-based Collaboration Services. *Information Systems Journal*, 2009.

[9] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. 't Hart. Enabling multi-tenancy: An industrial experience report. In *ICSM*, pages 1–8, 2010.

[10] F. Chong and G. Carraro. Architecture Strategies for Catching the Long Tail. 2006.

[11] A. Göbel. Anforderungen von Cloud-Anwendungen an Datenbanksysteme. In *Workshop Database as a Service*, 2010.

[12] D. Jacobs and S. Aulbach. Ruminations on Multi-Tenant Databases. In *BTW*, pages 514–521, 2007.

[13] S. Krompass, D. Gmach, A. Scholz, S. Seltzsam, and A. Kemper. Quality of Service Enabled Database Applications. In *ICSOC*, pages 215–226, 2006.

[14] S. Krompass, A. Scholz, M.-C. Albutiu, H. A. Kuno, J. L. Wiener, U. Dayal, and A. Kemper. Quality of Service-enabled Management of Database Workloads. *IEEE Data Eng. Bull.*, 31(1):20–27, 2008.

[15] T. Kwok and A. Mohindra. Resource Calculations with Constraints, and Placement of Tenants and Instances for Multi-tenant SaaS Applications. In *ICSOC*, pages 633–648. Springer-Verlag, 2008.

[16] W. Lang, S. Shankar, J. Patel, and A. Kalhan. Towards Multi-Tenant Performance SLOs. In *ICDE '12*, 2012.

[17] P. M. Nadkarni and C. Brandt. Data extraction and ad hoc query of an entity–attribute–value database. *Journal of American Medical Informatics Association*, 5(6):511–527, 1998.

[18] Pierre Audoin Consultants. Entry of the global players confirms Global SaaS Trends. 2012.

[19] B. Reinwald. Database support for multi-tenant applications. *IEEE Workshop on Information and Software as Services*, 1:2, 2010.

[20] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. Native support of multi-tenancy in RDBMS for software as a service. In *EDBT/ICDT*, pages 117–128. ACM, 2011.

[21] M. Seibold and A. Kemper. Database as a Service. *Datenbank-Spektrum*, 12(1):59–62, 2012.

[22] R. Sturm, W. Morris, and M. Jander. *Foundations of Service Level Management*. SAMS Publishing, Apr. 2000.