ORESTES: ein System für horizontal skalierbaren Zugriff auf Cloud-Datenbanken

Felix Gessert, Florian Bücklers

Universität Hamburg, Computer Science Department
gessert@informatik.uni-hamburg.de, fbuecklers@gmail.com

Art der Arbeit: Bachelorarbeit und Masterarbeit
Betreuer der Arbeit: Prof. Dr. Norbert Ritter

Abstract: Die Nutzbarkeit von Datenbanksystemen in Cloudumgebungen ist derzeit durch drei Probleme stark eingeschränkt: I) fehlende Mechanismen zur elastischen horizontalen Skalierung, II) hohe Netzwerklatenzen und III) Uneinheitlichkeit der Zugriffsprotokolle und -Schnittstellen. Um Database-as-a-Service Modelle attraktiv und umsetzbar zu machen, müssen diese Probleme gelöst werden. Wir stellen deshalb ORESTES (Objects RESTfully encapsulated in standard formats) vor. ORESTES ist ein auf REST/HTTP basierendes System für Datenbankzugriff, das in der Lage ist, Lese-Anfragen horizontal zu skalieren. Die Skalierbarkeit wird durch Web-Infrastruktur aus Web-Caches und Load-Balancern erzielt, unter Erhaltung der ACID Transaktionssemantik durch optimistische Validierung (I). ORESTES gestattet es, Web-Caches in beliebigen Topologien anzuordnen. Das Latenzproblem kann deshalb durch Web-Caches gelöst werden, die geographisch dicht am Client positioniert sind (II). Dies kann u.a. durch Web-Caches im Netzwerk des Clients oder die Nutzung verbreiteter Content Delivery Networks (CDNs) geschehen. Die REST/HTTP Schnittstelle ist so konzipiert, dass beliebige Datenmodelle und Kombinationen aus Persistenz APIs und Datenbanksystemen möglich sind (III). Die vergleichende Evaluation unserer Implementierung zeigt, dass ORESTES gegenüber existierenden Systemen eine signifikante Verbesserung der Latenz, Skalierbarkeit, Elastizität, Erweiterbarkeit und Schnittstellenoffenheit erreicht.

1 Motivation

Die Wissenschaft und Praxis von Datenbanksystemen befindet sich in einem der bisher radikalsten Umbrüche seiner Geschichte. Drei Paradigmenwechsel sind eindeutig identifizierbar: der zunehmende Wechsel von dem zuvor uneingeschränkt dominierenden relationalen Datenmodell zu domänenspezifischen, anforderungsgetriebenen Datenmodellen und Datenbanken („NoSQL“, „Polyglot Persistence“), die Ausrichtung vieler Datenbanken auf Technologien des Webs und seine Größenordnungen („Horizontal Scalability“, „Big Data“) und die häufige Verschiebung von Datenverarbeitung und –Speicherung auf Cloud Computing Plattformen (z.B. Amazon Web Services und Windows Azure) [CJP+11]. Um diesem Paradigmenwechsel technisch angemessen zu begegnen, sind viele Neuerungen notwendig. Wir analysieren die gewandelten Anforderungen und stellen das System ORESTES vor, das sie auf Basis existierender Technologien (Web-Infrastruktur und Datenbanksysteme) umsetzt.

Der Erfolg neuer NoSQL Systeme erklärt sich aus zwei Anforderungen – horizontaler Skalierbarkeit und einfachen Schnittstellen (cf. [Cat11]). Die Fähigkeit zur horizontalen Skalierbarkeit ermöglicht es, Datenbanksysteme auch für Szenarien mit besonders großem Datenvolumen und massive Workloads von hoher Parallelität einzusetzen. Die Beliebtheit alternativer Datenmodelle und einfacher Schnittstellen erklärt sich aus der erhöhten Produktivität der Anwendungsentwicklung und dem Wegfall des *Impedance Mismatch* [INW+09]. Ein Datenbanksystem sollte sich deshalb auch für den Einsatz mit anderen Datenbanksysteme eignen (*Polyglot Persistence*). Viele Anwendungen in Cloudumgebungen kommunizieren mit genutzten Services (zu denen Persistenz zählt) über statusloses HTTP. Die Unterstützung von Web Standards und Protokollen wird deshalb zunehmend zu einer zentralen Anforderung für Datenbanksysteme. Datenbanken, die als Service angeboten werden, benötigen die Möglichkeit, Leseanfragen zu skalieren. Außerdem müssen sie die geographische Verteilung von Datenbank und Applikation durch die Gewährleistung einer geringen Netzwerklatenz ermöglichen. Eine geringe Latenz ist die unabdingbare Voraussetzung, um hohen Durchsatz und die Reaktivität eines lokalen Datenbank-Deployments zu erreichen. Häufig wird mit einem Verweis auf das CAP Theorem (*Consistency, Availability, Partition Tolerance*) [GL02] ein Fehlen von Transaktionsmechanismen und starker Konsistenz entschuldigt. Wir sind jedoch der Überzeugung, dass Transaktionen eines der wichtigsten Ergebnisse der gesamten Datenbankforschung sind und als Option zur Verfügung stehen sollten. Die Anforderungen an ORESTES sind deshalb:

1. Elastische horizontale Skalierbarkeit
2. Erweiterbare und intuitive Datenbankschnittstelle
3. Umsetzung von Polyglot Persistence
4. Nutzung von Web Standards
5. Webbasiertes Monitoring, Deployment und Administrieren
6. Zugriff mit niedriger Netzwerklatenz
7. ACID Transaktionen und starke Konsistenz als Option

Diese Anforderungen erfüllt ORESTES durch die in Abbildung 1 dargestellte Architektur. Persistenz APIs (z.B. die Java Persistence API) bilden ihre Operationen auf die generische REST/HTTP Schnittstelle von ORESTES ab. Auf Netzwerkebene können durch die statuslose und cache-optimierte Natur der Kommunikation Web-Caching und Load-Balancing genutzt werden. Auf Serverseite bildet ein ORESTES-Server die REST Aufrufe auf ein zugrundeliegendes Datenbanksystem ab.

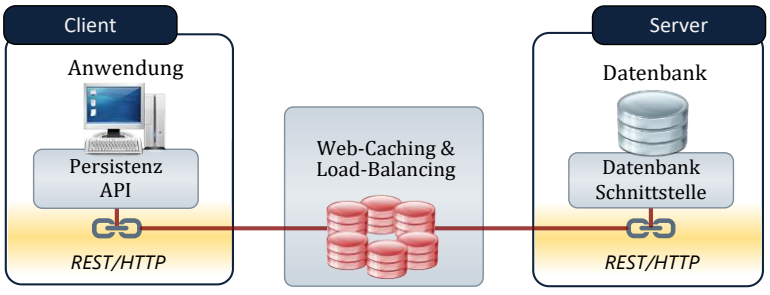

Abbildung 1: Architektur von ORESTES

Die Web-Cache Server agieren als asynchron aktualisierte Replikate des Datenbanksystems (*asynchrone Master-Slave Replikation*). Durch Load-Balancing kann die Anfrage-Last horizontal auf mehrere ORESTES-Server und Web-Caches skaliert werden. In der Nutzbarkeit von Web-Caching und Load-Balancing unterscheidet sich ORESTES fundamental von anderen Cloud Data Management, NoSQL- und DBaaS-Systemen mit REST Schnittstellen, die beide Mechanismen nicht unterstützen. Dies liegt in der Schwierigkeit begründet, das Caching-Modell von HTTP mit jenem eines Datenbanksystems in Einklang zu bringen. Ressourcen (also z.B. Datenbankobjekte) werden in HTTP für eine fest angegebene Caching-Dauer als aktuell betrachtet. Ad-hoc Invalidierung sind nur in wenigen Spezialfällen möglich. Um dennoch Konsistenz zu gewährleisten nutzt ORESTES zwei Mechanismen: optimistische Validierung und bloomfilterbasierte Cache-Kohärenz. Während ersterer ein wohlbekanntes Konzept der Datenbankforschung ist, handelt es sich bei zweiterem, um einen neuen Algorithmus, den wir zur Nutzung von Caches einführen, die auf Expiration basieren. Die optimistische Validierung basiert darauf, dass die Persistenz API des Clients die Versionsnummern aller gelesenen Objekte als Read-Set hält und beim Transaktionscommit an den Server überträgt. Dies erlaubt es zu validieren, dass keine veralteten Objekte gelesen wurden und keine Konflikte zu nebenläufigen Transaktionen aufgetreten sind (*Backward-Oriented Optimistic Concurrency Control*).

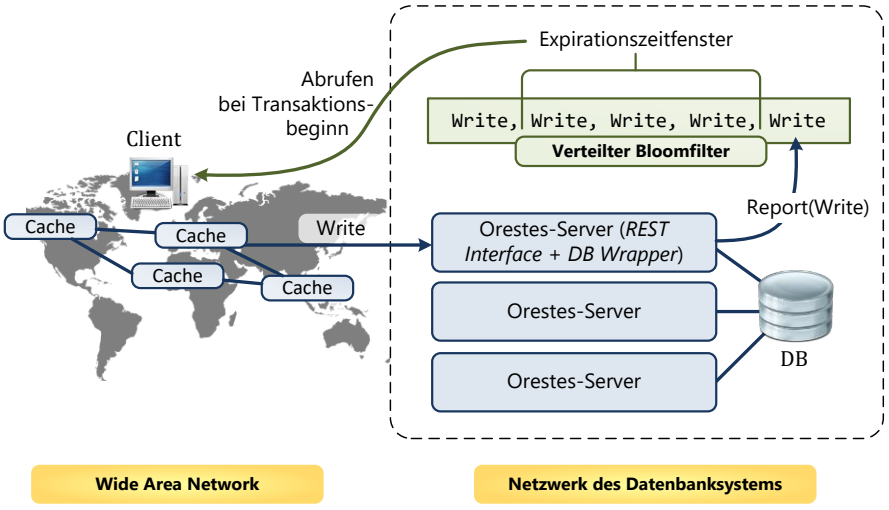

Abbildung 2: Bloomfilterbasierte Cache-Kohärenz in ORESTES

Die von uns eingeführte bloomfilterbasierte Cache-Kohärenz ist ein Algorithmus, der die Cache-Kohärenz von Web-Caches, die eigentlich nach dem Prinzip *Eventual Consistency* arbeiten, zu einem anpassbaren Parameter macht. Wie in Abbildung 2 dargestellt, werden dazu Schreibvorgänge in einem (verteilten) Counting Bloomfilter registriert, einer probabilistischen, sehr kompakten Mengenrepräsentation. Der Bloomfilter enthält für die Periode einer Caching-Dauer alle geänderten Objekte. Bei Transaktionsbeginn wird er optimiert zum Client übertragen, der anschließend für alle Objekte durch einen $O(1)$ Hash-Lookup feststellen kann, ob sie potentiell veraltet sind (d.h. im Bloomfilter enthalten sind) und direkt vom Server geladen werden müssen (*Revalidierung*).

2 Evaluation und Ausblick

Unsere ORESTES Implementierung unterstützt derzeit zwei Persistenz-APIs: Java Data Objects (JDO) und eine von uns entworfene Portierung der Java Persistence API auf JavaScript (JSPA). Als serverseitige Datenbankbackends werden die beiden objektorientierten Datenbanken Versant Object Database (VOD) und db4o und sowie der NoSQL Key-Value Store Redis unterstützt. Das generische ORESTES Framework auf dem die Anbindungen basieren, stellt Funktionen wie Transaktionsverwaltung und ein umfassendes Browser-Interface mit integrierter Entwicklungsumgebung bereit. Abbildung 3 zeigt die in einer Cloudumgebung ermittelten Performancevorteile von ORESTES anhand eines Vergleichs von VOD mit seiner nativen TCP-Schnittstelle und VOD als Backend von ORESTES. Der Benchmark ermittelt die durchschnittliche Ausführungsdauer von 50 dedizierten Clientsystemen unter Operationen wie Lesen, Schreiben und Querying.

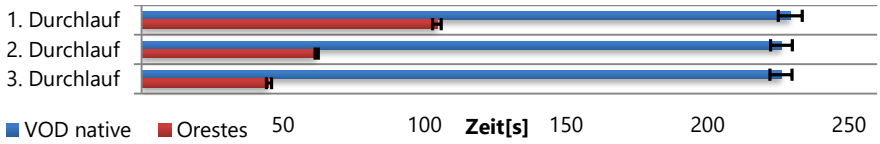

Abbildung 3: Ergebnisse der Evaluation von ORESTES in einer Cloudumgebung

Wir wollen unseren Ansatz künftig in viele Richtungen erweitern, vor allem bezüglich der Nutzung als skalierbaren Cloud Service. Die automatische Allokation von Ressourcen wie Caching-Servern und Sicherstellung von Quality-of-Service Garantien z.B. für die Quantile der Latenzzeiten sind dabei wichtige Punkte. Durch eine Erweiterung von ORESTES auf andere Datenmodelle (NoSQL-) Datenmodelle hoffen wir eine gemeinsame Basis für die sehr unterschiedlichen Klassen von Datenbanken schaffen zu können.

Literaturverzeichnis

[Fie00] Fielding, R. T.: Architectural styles and the design of network-based software architectures, *University of California*, 2000.

[WV02] Weikum, G.; Vossen, G.: Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann Pub, 2002.

[Här05] Härder, T.: Caching over the entire user-to-data path in the internet, *Data Management in a Connected World*, pp. 150–170, 2005.

[Cat11] Cattell, T.: Scalable sql and nosql data stores, *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[CJP+11] Curino, C. A.; Jones, E. P. C.; Popa, R. A.; Malviya, N.; Wu, E.; Madden, S. R.; Balakrishnan, H.; Zeldovich, N.: Relational cloud: A database-as-a-service for the cloud, *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, pp. 235–241, Pacific Grove, CA, 2011.

[GL02] Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 2002.