Messbarkeit und Beeinflussung von Eventual-Consistency in verteilten Datenspeichersystemen[1]

David Bermbach[2]

Abstract: Cloudspeicherdienste und NoSQL-Systeme, die sich zunehmend größerer Beliebtheit erfreuen, bieten meist weder transaktionale Features noch strikte Konsistenzgarantien. Stattdessen wird mit Eventual-Consistency lediglich garantiert, dass alle Schreiboperationen irgendwann – jedoch zu einem undefinierten Zeitpunkt – auf allen Replika ausgeführt werden. Die Unsicherheit, wann dies passiert, stellt dabei Anwendungsentwickler, die ein solches System nutzen möchten, vor große Schwierigkeiten, da es jederzeit möglich ist, dass veraltete Daten gelesen werden oder parallele Updates zu weitergehenden Problemen führen.

Mit dieser Arbeit wird erstmals ermöglicht, durch Experimente und Simulationen Wissen über den Grad der Inkonsistenz zu gewinnen und mit ebenfalls vorgestellten Verfahren auf Basis dieses Wissens Inkonsistenzen in der Anwendungsschicht aufzulösen oder sogar durch eine Middlewareschicht zusätzliche Konsistenzgarantien zu geben.

1 Motivation

Hohe Verfügbarkeit und Fehlertoleranz aber auch geringe Antwortzeiten sind für die meisten Anwendungen von größter Wichtigkeit. Typischerweise erreicht man dies durch Replikation, d.h. man führt mehrere lose gekoppelte Instanzen dieser Anwendung parallel auf verschiedenen Maschinen aus. Je mehr Instanzen dabei verwendet werden und je größer deren Unabhängigkeit und geographische Verteilung, desto größer ist auch die Wahrscheinlichkeit, dass stets zumindest eine der parallelen Instanzen verfügbar ist. Für zustandslose Systeme, wie sie bspw. in der Anwendungsschicht vorherrschen, kann dies als ein weitgehend gelöstes Problem betrachtet werden, z.B. [LV11]. Für zustandsbehaftete Systeme hingegen, hierunter fallen alle Arten von Datenbank- oder Speichersystemen, ist dies alles andere als trivial, da dabei sichergestellt werden muss, dass bei Schreiboperationen alle Replika aktualisiert werden. In diesem Kontext müssen u.A. die Konsistenztradeoffs des CAP-Theorems [Br00] und des PACELC-Modells [Ab12] beachtet werden.

Da insbesondere im Webkontext aber auch in anderen Bereichen oftmals Verfügbarkeit, Skalierbarkeit und geringe Latenzen als wichtiger als strikte Konsistenzgarantien und transaktionale Interfaces erachtet werden, hat dies dazugeführt, dass es neben den klassischen relationalen Datenbanken mit transaktionalen Features und strikten Konsistenzgarantien heutzutage ein Vielzahl von Speicherlösungen gibt, die bewusst auf reduzierte Konsistenzgarantien setzen, um ihre Verfügbarkeits-, Skalierbarkeits oder Performanceziele erreichen

[1] Englischer Titel der Dissertation: “Benchmarking Eventually Consistent Distributed Storage Systems”

[2] Technische Universität Berlin, Information Systems Engineering Research Group, Einsteinufer 17, 10587 Berlin, db@ise.tu-berlin.de

zu können. Diese Systeme werden, da sie typischerweise nicht SQL als Anfragesprache nutzen, als NoSQL (*N*ot *O*nly *SQL*) bezeichnet. In diese Kategorie fallen neben Opensourcesystemen wie bspw. Cassandra [LM10] oder dem Hadoopstack[2] insbesondere auch nahezu alle dem Autor bekannten Cloudspeicherdienste. Darunter sind auch so populäre Dienste wie Amazon S3[3], das bspw. Dropbox[4] zu Grunde liegt und bereits im April 2013 bei bis dato exponentiellem Wachstum mehr als $2 * 10^{12}$ Objekte gespeichert und gleichzeitig 1,1 Millionen Anfragen pro Sekunde beantwortet hatte [Ba13b]. Allein hierin sieht man die große wirtschaftliche Bedeutung und Verbreitung dieser Systeme und Dienste.

2 Problemstellung

Während Anwendungen die gute Performance sowie hohe Verfügbarkeit und Skalierbarkeit der ihnen zu Grunde liegenden neuen Speichersysteme zu Gute kommen, führen jedoch gleichzeitig die reduzierten Konsistenzgarantien des Speichersystems zu massiven Problemen in der Anwendungsschicht: Es gibt bspw. keine Garantie dafür, dass ein gelesener Wert aktuell ist oder auch nur dass er nicht älter als der zuletzt gelesene Wert ist. Die einzige Garantie, die Eventual-Consistency gibt, ist, dass nach einer hinreichend langen Zeit ohne weitere Schreibrequests oder Fehler im System alle Replika in Richtung einer gemeinsamen Datenversion kovergieren. Dies bedeutet insbesondere auch, dass Anwendungen die Konfliktauflösung aufgezwungen wird, obwohl diese Veranwortlichkeit eigentlich entsprechend der Prinzipien von Schichtenarchitekturen und Replikationstransparenz in der Datenspeicherschicht liegen sollte.

Gleichzeitig bedeutet eine schwache Garantie jedoch nicht, dass die Probleme für jede Anwendungen immer real beobachtbar sind – dies hängt nicht zuletzt vom Zusammenspiel von Anwendungsworkload und konkretem Speichersystem ab. Da die Behandlung von Inkonsistenzen – sofern überhaupt möglich – technisch aufwändig und somit teuer ist, stellt sich aus ökonomischer Sicht die Frage, wann es sich überhaupt lohnt, entsprechende Mechanismen zu implementieren, bzw. wann es möglicherweise günstiger ist, monetäre Kompensationsmechanismen zu nutzen.

Ein Beispiel hierfür wäre ein Onlinehändler, bei dem zwei Kunden gleichzeitig das letzte Buch, was auf Lager ist, bestellen. Beide sehen in ihrem Webclient, dass das Buch verfügbar ist, und ihre jeweilige Bestellung wird von zwei verschiedenen Datenbankreplika angenommen. Zwangsläufig kann nun nur einer der beiden Kunden beliefert werden, so dass der jeweils andere verärgert wird. In solchen Fällen ist es nicht unüblich, der Verärgerung vorzubeugen, indem man dem Kunden als Form der monetären Kompensation einen Warengutschein schenkt. Die Alternative hierzu wäre, dass bei Annahme einer Bestellung über eine Form von Consensusprotokoll sichergestellt wird, dass nur ein Kunde gleichzeitig das letzte Buch bestellen kann. Ob es sich nun aus ökonomischer Sicht lohnt, technische Mittel wie bspw. ein Consensusprotokoll zu verwenden, oder ob eine monetäre Kompensation günstiger ist, hängt damit von den folgenden vier Fragen ab:

[2] hadoop.apache.org
[3] aws.amazon.com/s3
[4] dropbox.com

1. Wie oft kommt es vor, dass der Lagerbestand für ein Produkt so niedrig ist?
2. Wie schnell synchronisiert das System seine Replika und stellt damit sicher, dass nur aktuelle Werte gelesen werden?
3. Wie häufig kommt es vor, dass es innerhalb des dadurch definierten Zeitfensters überhaupt mehr als einen Kauf des gleichen Produkts gibt?
4. Was kostet die monetäre Kompensation eines Kunden?

Die Fragen 1 und 4 lassen sich leicht durch die entsprechenden Unternehmensabteilungen bzw. durch historische Daten beantworten. Frage 3 lässt sich (mit Hilfe von Frage 2) beantworten, wenn man sich Workloadstatistiken des Onlinehändlers anschaut. Frage 2 hingegen ist ohne die diesem Artikel zu Grunde liegende Dissertation [Be14] nicht beantwortbar.

3 Beiträge der Dissertation

Neben der Inkonsistenzart aus dem Beispiel des Onlinehändlers gibt es noch einige weitere Inkonsistenzarten, die darüberliegende Anwendungen auf jeweils unterschiedliche Art und Weise beeinflussen. Je nach Art können dabei entstehende Konflikte ohne die Antwort zu obiger Frage 2 gar nicht oder nur sehr aufwändig aufgelöst werden. Die Dissertation, auf der dieser Artikel beruht, versucht daher, diese Frage für gegebene Speichersysteme möglichst allgemeingültig und für alle diese Konfliktarten zu beantworten.

Dies erfolgt einerseits auf Basis neuartiger Benchmarks aber auch mit Hilfe ebenso neu entwickelter Simulationsverfahren. Die Arbeit geht jedoch noch einen Schritt weiter und zeigt auch auf, wie das dadurch gewonnene Wissen genutzt werden kann, um Inkonsistenzen aus einer Anwendungssicht zu behandeln. Damit bietet die Dissertation nicht nur Anwendungsentwicklern die Möglichkeit, besser mit Inkonsistenzen umzugehen, sondern sie liefert auch das notwendige Handwerkszeug, um verteilte Speichersysteme, die nur Eventual-Consistency garantieren, in ihrem tatsächlichen Konsistenzverhalten zu untersuchen und Einflussfaktoren damit besser oder überhaupt erstmals verstehen zu können.

Daher beinhaltet die Dissertation die folgenden Beiträge, die über bestehende Ansätze hinausgehen:

1. *Aussagekräftige Konsistenzmetriken*: Die Arbeit entwickelt auf Basis von Literaturrecherche eine Menge an Konsistenzmetriken, die es ermöglicht, Konsistenzverhalten für alle Konsistenzperspektiven, -dimensionen und -modelle zu beschreiben. Dabei sind Grundanforderungen, dass diese Metriken präzise sind, auf überflüssige Aggregationen verzichten und direkte Aussagekraft für mindestens eine der potentiellen Zielgruppen (Anwendungsentwickler, Datenbankentwickler oder Forscher) haben.
2. *Modellierung und Simulation von Konsistenzverhalten*: Anschließend werden die Haupteinflussfaktoren auf Konsistenzverhalten identifiziert und im Modell eines

verteilten Speichersystems berücksichtigt. Aus Basis dieses Modells werden danach zwei verschiedene Simulationsverfahren vorgestellt, die mit jeweils unterschiedlicher Nutzungskomplexität das Konsistenzverhalten des modellierten Speichersystems approximieren können. Dabei werden die zuvor entwickelten Konsistenzmetriken zur Beschreibung der Ergebnisse benutzt.

3. *Benchmarking von Konsistenzverhalten*: Da Simulationen zwar kostengünstig und schnell sind, aber nie genauer als das zugrundeliegende Modell und die Inputdaten sein können, entwickelt die Arbeit anschließend Messverfahren für die verschiedenen Konsistenzarten, die im Rahmen von Benchmarks das Konsistenzverhalten des jeweiligen Systems experimentell untersuchen können.

4. *Behandlung von Inkonsistenzen*: Um den Nutzen der so gewonnenen Informationen zu demonstrieren, diskutiert die Dissertation abschließend, wie diese Informationen im konkreten Anwendungsszenario eines Webshops genutzt werden können. Dies kann in weiten Teilen nicht usecase-unabhängig beschrieben werden, da für die Auflösung einiger Inkonsistenzarten neben dem Wissen über Konsistenzverhalten auch anwendungsspezifisches Wissen notwendig ist. Für einen Teil der usecase-unabhängigen Konfliktauflösungsverfahren wird darüberhinaus jedoch auch noch eine Middlewarekomponente vorgestellt, die zwischen Anwendung und Speichersystem liegend für die Anwendung Konsistenzgarantien geben kann, die das Speichersystem selbst nicht geben kann.

Um diese Beiträge zu stützen, beinhaltet die Arbeit auch die prototypische Entwicklung aller vorgestellten Simulations- und Benchmarkingverfahren sowie der Middlewarekomponente. Die jeweiligen Softwareartefakte werden genutzt, um über eine Vielzahl von Messungen und Experimenten das Konsistenzverhalten von Amazon S3, Apache Cassandra, MongoDB[5] und Amazon DynamoDB[6] exemplarisch zu untersuchen, sowie die Simulationsansätze und das korrekte Verhalten der Middlewarekomponente zu demonstrieren.

4 Verwandte Arbeiten

Da das Themenfeld noch sehr neu ist und die Probleme erst seit Mitte der 2000er für größere Gruppen von Anwendungsentwicklern relevant geworden sind, gibt es relativ wenige direkt verwandte Arbeiten. Die, die es für die einzelnen Bereiche gibt, sind entweder erst später publiziert worden als die jeweils der Dissertation zu Grund liegenden Publikationen und stellen alternative Ansätze zu Einzelaspekten dar ([GLS11, Ra12, ZK12, Ba13a]). Alternativ sind diese Arbeiten auf unterschiedliche Art und Weise in ihrer Anwendbarkeit eingeschränkt, indem sie bspw. nur für ein konkretes Speichersystem nutzbar sind oder Inkonsistenzen nur unter bestimmten Annahmen detektieren können ([Wa11, An10, Pa11, Ba12]).

In jedem Fall ist die diesem Artikel zu Grunde liegende Dissertation die erste Arbeit, die das Zusammenspiel von Simulation und Benchmarking von Konsistenzverhalten sowie

[5] mongodb.org
[6] aws.amazon.com/dynamodb

die Nutzung derer Ergebnisse beschreibt. Weitere verwandte Arbeiten sind nicht direkt verwandt und stellen somit keine unmittelbaren Alternativansätze zu den Beiträgen dieser Arbeit dar (z.B. [KMF10, Co10]).

5 Untersuchung von Konsistenzverhalten

In diesem Abschnitt soll als Beispiel für die Beiträge der Arbeit eines der Benchmarking-verfahren vorgestellt werden. Es versucht für Eventual-Consistency die Frage zu beantworten, wie lange es nach einer Schreiboperation eigentlich dauert, bis Anwendungsclients stets den zuletzt geschriebenen Wert lesen, wie groß also das Inkonsistenzfenster, die sogenannte Staleness, ist. Dies entspricht auch in etwa der Frage 2 aus Abschnitt 2 und beantwortet bezogen auf Eventual-Consistency die plakative Frage nach “How soon is eventual?”.

5.1 Staleness als Metrik

Die Staleness-Metrik, die hier verwendet werden soll, ist die sogenannte t-Visibility, die bezogen auf Staleness gewissermaßen das Worst-Case-Verhalten beschreibt; t-Visibility ist dabei eine Dichtefunktion, die beschreibt, mit welcher Häufigkeit welches Inkonsistenzfenster für Anwendungsclients beobachtbar ist. Das Inkonsistenzfenster ist dabei für Systeme mit Eventual-Consistency definiert als das Zeitintervall zwischen dem Start einer Schreiboperation und dem frühestmöglichen Zeitpunkt, für den gilt, dass eine Leseoperation garantiert immer die aktuelle Datenversion liest.

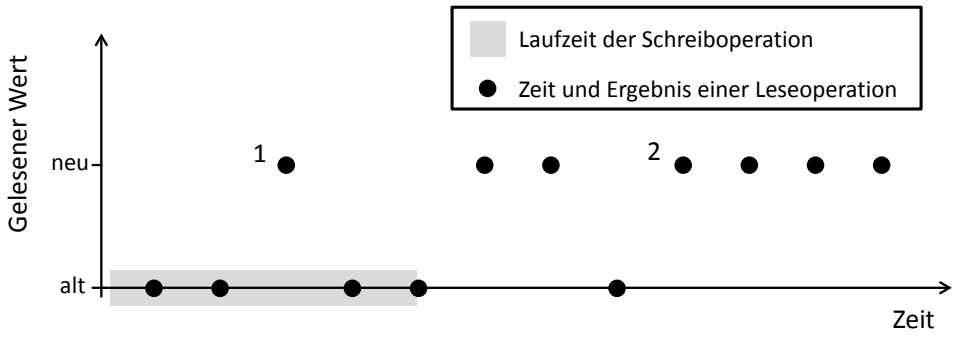

Abb. 1: Beispiel für die Bestimmung von Staleness

5.2 Messung von Staleness

Aus der Metrik t-Visibility leitet sich schon unmittelbar der Grundansatz des Messverfahrens ab, nämlich einen Wert zu schreiben und dann über eine große Menge an Leseoperationen zu beobachten, ab wann der überschriebene Wert beim Lesen nicht mehr zurückgegeben wird. Hierbei ist es wichtig zu beachten, dass der Zeitpunkt, an dem das

erste Mal der neue Wert gelesen wird (Punkt 1 in Abb. 1), in den allermeisten Fällen nicht mit dem Zeitpunkt identisch ist, ab dem garantiert nur noch der neue Wert zurückgegeben wird (Punkt 2 in Abb. 1), aber nur letzterer für die Messung von t-Visibility relevant ist. Demzufolge ergibt sich für einen einzelnen t-Visibility-Messwert im Beispiel aus Abb. 1 das Intervall, das durch den Beginn des grau hinterlegten Bereiches und Punkt 2 definiert ist. Wenn man dies hinreichend oft wiederholt (d.h. die Sequenz aus Schreiben und sehr oft Lesen), erhält man eine Stichprobe, aus der sich dann die Dichtefunktion von t-Visibility berechnen lässt. Oft ist es aber auch interessant, die Messwerte in chronologischer Reihenfolge zu betrachten, um sicherzustellen, dass man durch die Aggregation keine Periodizitäten oder Trends verschleiert.

Was ebenfalls beim Betrachten von Abb. 1 auffällt, ist, dass die Messgenauigkeit durch den Abstand der schwarzen Punkte, d.h. durch den Abstand zweier Leseoperationen, und durch die Größe der schwarzen Punkte, d.h. die Leselatenz, bestimmt wird. Aus diesem Grund ist es wichtig, die Messung verteilt durchzuführen, indem man mehrere Maschinen gleichzeitig lesen lässt. Hierbei entsteht zwar das Problem der Uhrensynchronisation, was die Messgenauigkeit ebenfalls beeinträchtigt, aber dennoch deutlich genauere Messungen als mit einer Messmaschine ermöglicht. Dabei ist es jedoch wichtig, diese Maschinen möglichst nahe zu den Replika zu positionieren, um die Leselatenz zu minimieren. Bei geografisch verteilten Replika ist dies folglich ebenfalls nur über eine verteilte Messung möglich. Die Berechnung der Metrik ändert sich durch die Verteilung jedoch nicht – die Beispielpunkte in Abb. 1 liegen dann nur verteilt auf unterschiedliche Maschinen als Messwert vor.

Neben diesen Punkten gibt es noch ein weiteres Argument pro Verteilung: Das Konzept von “sticky sessions”, was häufig verwendet wird, sorgt dafür, dass ein Clientrequest immer zum gleichen Server geroutet wird. Wenn man auch für Speichersysteme, die einen solchen Load Balancer nutzen, Staleness messen möchte, benötigt man mindestens genauso viele verteilte Messclients wie Replika. In unseren Experimenten hat es sich bewährt, für ein Speichersystem mit 3 Replika eine Anzahl von 12 Messclients nur zum Lesen (Reader) sowie je eine weitere dedizierte Maschine zum Schreiben (Writer) und zum Sammeln der Messergebnisse zu verwenden.

Um zwischen den Messungen nicht unnötig viel Zeit mit warten zu verbringen und gleichzeitig auch identifizieren zu können, ob womöglich gerade sogar der Vorgänger des “alten” Wertes gelesen wurde, schreibt der Writer eine Kombination aus einer Versionsnummer und seinem lokalen Timestamp. Die Reader können dann entsprechend über die Versionsnummer das Alter in Versionen und den t-Visibility-Messwert über den Timestamp bestimmen.

5.3 Staleness-Messwerte bei Amazon S3

Als ein Beispiel, wie t-Visibility-Messwerte für reale Systeme aussehen, sollen hier unsere Experimente mit Amazon S3 im August 2011 dienen, da diese besonders deutlich illustrieren, wie wichtig es ist Konsistenzverhalten zu untersuchen. Amazon S3 ist ein sogenannter

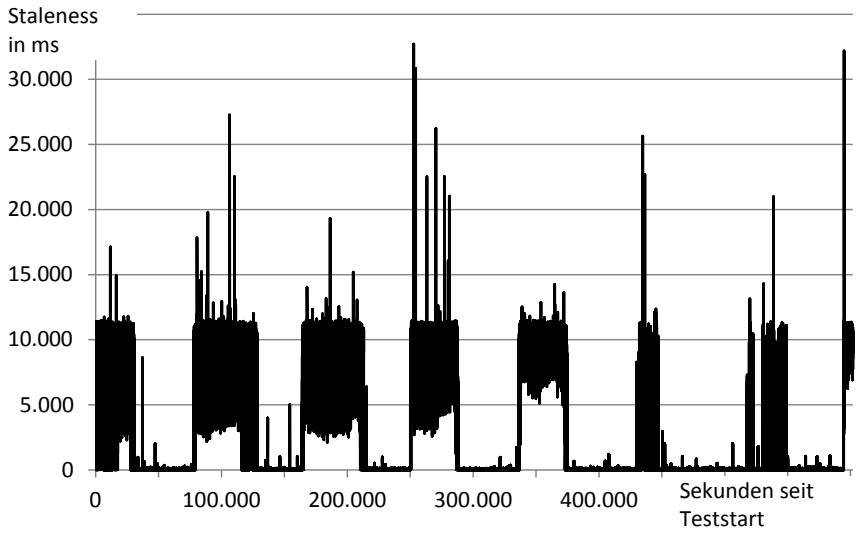

Abb. 2: Staleness-Messwerte für Amazon S3 (2011)

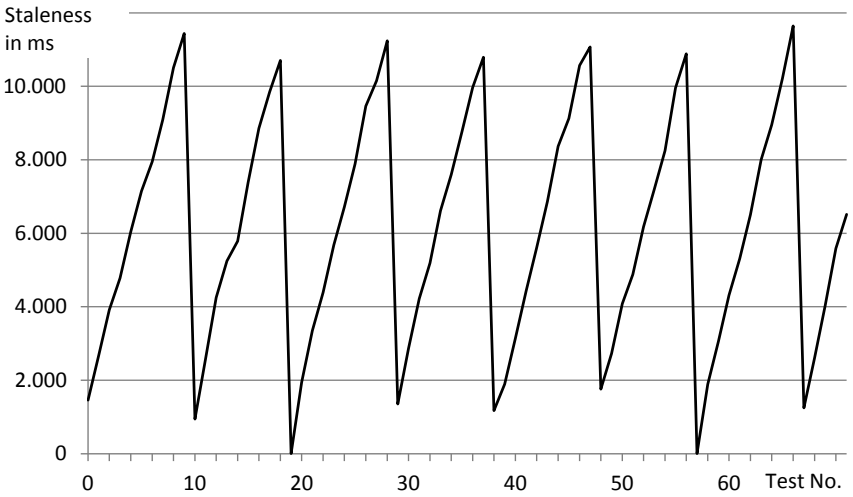

Abb. 3: Staleness-Messwerte für Amazon S3 (2011), Tagverhalten

Key-Value-Store, d.h. eigentlich ein verteilter Hashtable jedoch mit sogenannten Buckets, die eine hierarchische Struktur ermöglichen, der mit einem CRUD-Interface eher begrenzte Querymöglichkeiten aufweist. S3 garantiert lediglich Eventual-Consistency, aber wir wollten untersuchen "wie inkonsistent" die Daten denn eigentlich sind. Dafür erstellten wir ein Bucket und eine Testdatei in der Amazon-Region eu-west und starteten auf EC2[7] m1.small-Instanzen[8] in der gleichen Region – im oben beschriebenen (12,1,1)-Setup. Als Intervall zwischen zwei Schreiboperationen stellten wir 10 Sekunden ein, während die Reader einfach möglichst viele Leseoperationen erzeugten.

Abb. 2 zeigt die Ergebnisse dieses einwöchigen Tests (Start Montag 29.8.2011, später Vormittag) als Zeitreihe der Einzelmesswerte. Entgegen der Erwartung von zufällig verteilten Werten, lassen sich klare Periodizitäten erkennen, die jeweils vollkommen unterschiedliches Konsistenzverhalten für Tag und Nacht zeigen. Noch kurioser wird es, wenn man sich die Messergebnisse während des Tages genauer anschaut, wo man weitere Periodizitäten erkennen konnte. Abb. 3 zeigt das Tagverhalten von S3 im Jahr 2011 – eine Art Sägezahnmuster. In Verbindung mit weiteren (allesamt reproduzierbaren) Benchmarks liegt die Vermutung nahe, dass Amazon zu dem Zeitpunkt Updates entweder nach Ablauf eines fixen Zeitraums oder alternativ nach Eintreffen eines zweiten Requests für das gleiche Bucket propagiert hat. Infolgedessen hatte der Autor im Folgenden mehrfach Kontakt mit Amazon und konnte im Zeitraum 2011 bis 2013 in Reaktion auf die Interaktion jeweils massive Veränderungen im Konsistenzverhalten von S3 beobachten – offensichtlich wurde die Implementierung von S3 angepasst, um bessere Konsistenzergebnisse erzielen zu können.

6 Zusammenfassung

In modernen Anwendungen mit ihren Anforderungen an Skalierbarkeit, Verfügbarkeit und Performance ist es fast undenkbar, Datenbanken und Speichersysteme zu verwenden, die strikte Konsistenz garantieren. Die Nutzung von Systemen, die nur Eventual-Consistency garantieren, kommt jedoch mit eigenen Problemen, da dort auf einmal der Anwendungsentwickler dafür verantwortlich ist, Inkonsistenzen und Konflikte aufzulösen, was er nur schwer oder sogar gar nicht kann.

In der diesem Artikel zu Grunde liegenden Dissertation [Be14] wurden daher nach einer detaillierten Diskussion von Grundlagen und verwandten Arbeiten zunächst Anforderungen an Konsistenzmetriken aus der Literatur abgeleitet und eine Menge an entsprechenden Metriken entwickelt. Anschließend wurden die Haupteinflussfaktoren auf Konsistenzverhalten identifiziert und im Modell eines Speichersystems berücksichtigt, auf Basis dessen dann zwei Simulationsansätze für diverse Konsistenzmetriken entwickelt wurden. Als dritter Beitrag wurden anschließend für jede der Konsistenzmetriken Messverfahren vorgestellt.

[7] aws.amazon.com/ec2
[8] EC2 ist der Amazondienst, der virtuelle Maschinen zur Verfügung stellt.

Im Anschluss daran wurden die Anwendbarkeit und die Anwendung der vorgestellten Ansätze in Bezug auf die Dimensionen Proof-of-Concept, Korrektheit und Relevanz demonstriert. Dazu wurden zunächst die Implementierungen aller Beiträge vorgestellt und anschließend die Ergebnisse einer Vielzahl von Simulationsläufen und Benchmarks präsentiert und miteinander verglichen. Final wurde dann der Nutzen der gewonnenen Informationen im Rahmen eines Usecases und der neuentwickelten Middlewarekomponente aufgezeigt.

In diesem Artikel konnte aus Platzgründen nur ein grober Überblick über die Beiträge der zu Grunde liegenden Dissertation gegeben werden. Exemplarisch wurden jedoch eine Metrik und das dazugehörige Benchmarkingverfahren etwas detaillierter vorgestellt sowie ein Beispielmessergebniss präsentiert. Für alle Details und die übrigen Forschungsbeiträge sei an dieser Stelle auf die vollständige Dissertation [Be14] oder die ihr zu Grunde liegenden Publikationen verwiesen.

Literaturverzeichnis

[Ab12] Abadi, Daniel: Consistency Tradeoffs in Modern Distributed Database System Design: CAP is Only Part of the Story. IEEE Computer, 45(2):37–42, Februar 2012.

[An10] Anderson, Eric; Li, Xiaozhou; Shah, Mehul A.; Tucek, Joseph; Wylie, Jay J.: What Consistency Does Your Key-value Store Actually Provide? In: Proceedings of the 6th Workshop on Hot Topics in System Dependability (HOTDEP). HotDep'10, USENIX Association, Berkeley, CA, USA, S. 1–16, 2010.

[Ba12] Bailis, Peter; Venkataraman, Shivaram; Franklin, Michael J.; Hellerstein, Joseph M.; Stoica, Ion: Probabilistically Bounded Staleness for Practical Partial Quorums. Proceedings of the VLDB Endowment, 5(8):776–787, April 2012.

[Ba13a] Bailis, Peter; Ghodsi, Ali; Hellerstein, Joseph M.; Stoica, Ion: Bolt-on Causal Consistency. In: Proceedings of the 33rd International Conference on Management of Data (SIGMOD). SIGMOD '13, ACM, New York, NY, USA, S. 761–772, 2013.

[Ba13b] Bar, Jeff: Amazon S3 Two Trillion Objects, 1.1 Million Requests / Second. `https://aws.amazon.com/de/blogs/aws/amazon-s3-two-trillion-objects-11-million-requests-second/` (abgerufen am 11.02.2015), 2013.

[Be14] Bermbach, David: Benchmarking Eventually Consistent Distributed Storage Systems. Dissertation, Karlsruher Institut für Technologie, Februar 2014. KIT Scientific Publishing.

[Br00] Brewer, Eric: PODC Keynote. `http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf` (accessed Jun 27,2013), 2000.

[Co10] Cooper, Brian F.; Silberstein, Adam; Tam, Erwin; Ramakrishnan, Raghu; Sears, Russell: Benchmarking Cloud Serving Systems with YCSB. In: Proceedings of the 1st Symposium on Cloud Computing (SOCC). SOCC '10, ACM, New York, NY, USA, S. 143–154, 2010.

[GLS11] Golab, Wojciech; Li, Xiaozhou; Shah, Mehul A.: Analyzing Consistency Properties for Fun and Profit. In: Proceedings of the 30th Symposium on Principles of Distributed Computing (PODC). PODC '11, ACM, New York, NY, USA, S. 197–206, 2011.

[KMF10] Klems, Markus; Menzel, Michael; Fischer, Robin: Consistency Benchmarking: Evaluating the Consistency Behavior of Middleware Services in the Cloud. In (Maglio, Paul; Weske, Mathias; Yang, Jian; Fantinato, Marcelo, Hrsg.): Service-Oriented Computing, Jgg. 6470 in Lecture Notes in Computer Science, S. 627–634. Springer Berlin Heidelberg, 2010.

[LM10] Lakshman, Avinash; Malik, Prashant: Cassandra: A Decentralized Structured Storage System. SIGOPS Operating Systems Review, 44(2):35–40, April 2010.

[LV11] Li, Han; Venugopal, Srikumar: Using Reinforcement Learning for Controlling an Elastic Web Application Hosting Platform. In: Proceedings of the 8th International Conference on Autonomic Computing (ICAC). ICAC '11, ACM, New York, NY, USA, S. 205–208, 2011.

[Pa11] Patil, Swapnil; Polte, Milo; Ren, Kai; Tantisiriroj, Wittawat; Xiao, Lin; López, Julio; Gibson, Garth; Fuchs, Adam; Rinaldi, Billie: YCSB++: Benchmarking and Performance Debugging Advanced Features in Scalable Table Stores. In: Proceedings of the 2nd Symposium on Cloud Computing (SOCC). SOCC '11, ACM, New York, NY, USA, S. 9:1–9:14, 2011.

[Ra12] Rahman, Muntasir Raihan; Golab, Wojciech; AuYoung, Alvin; Keeton, Kimberly; Wylie, Jay J.: Toward a Principled Framework for Benchmarking Consistency. In: Proceedings of the 8th Conference on Hot Topics in System Dependability (HOTDEP). HotDep'12, USENIX Association, Berkeley, CA, USA, S. 8–8, 2012.

[Wa11] Wada, Hiroshi; Fekete, Alan; Zhao, Liang; Lee, Kevin; Liu, Anna: Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: the Consumers' Perspective. In: Proceedings of the 5th Conference on Innovative Data Systems Research (CIDR). S. 134–143, January 2011.

[ZK12] Zellag, Kamal; Kemme, Bettina: How Consistent is Your Cloud Application? In: Proceedings of the 3rd Symposium on Cloud Computing (SOCC). SOCC '12, ACM, New York, NY, USA, S. 6:1–6:14, 2012.

David Bermbach wurde am 11. Juni 1984 in Mainz geboren. Nach seinem Abitur im Jahr 2004 und Zivildienst in Wolfenbüttel, studierte er von 2005 bis 2010 Wirtschaftsingenieurwesen am Karlsruher Institut für Technologie (KIT), wo er sich jedoch schon früh auf Informatikinhalte spezialisierte und diese auch in diversen nationalen und internationalen Industriepraktika vertiefen konnte. Nach Abschluss seines Studiums, arbeitete er als wissenschaftlicher Mitarbeiter am KIT sowie als Dozent an der DHBW Karlsruhe. Am 10. Februar 2014 wurde er am KIT mit Auszeichnung zum Dr.-Ing. promoviert.

Seit Juni 2014 arbeitet er als Postdoc in der Gruppe von Prof. Dr. Stefan Tai an der TU Berlin. Seine Publikationen wurden mit je einem Best-Paper-Award und einem Best-Paper-Runner-Up-Award ausgezeichnet.