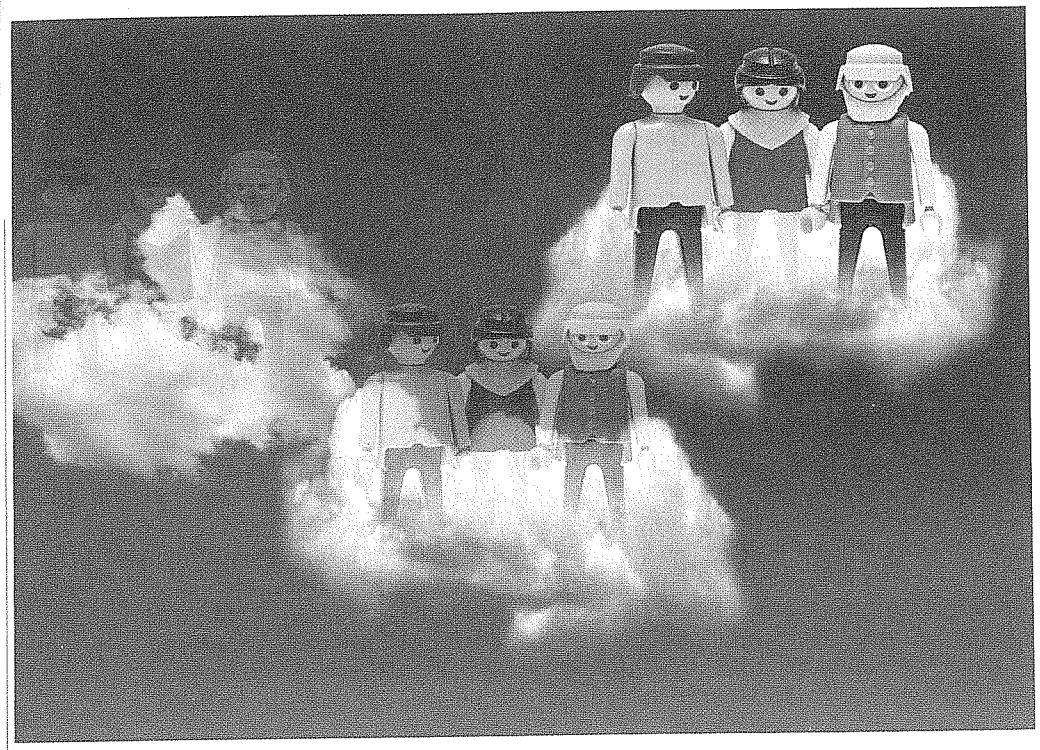


Speicherdienste in der Cloud

Sägende Konsistenz

David Bermbach, Stefan Tai

Wer Daten verteilt in der Cloud speichert, muss sich darauf einstellen, dass nicht alle Kopien jederzeit identisch sind. Untersuchungen von Amazons S3 zeigten, dass die Konsistenz dort unter anderem von der Tageszeit abhängt.

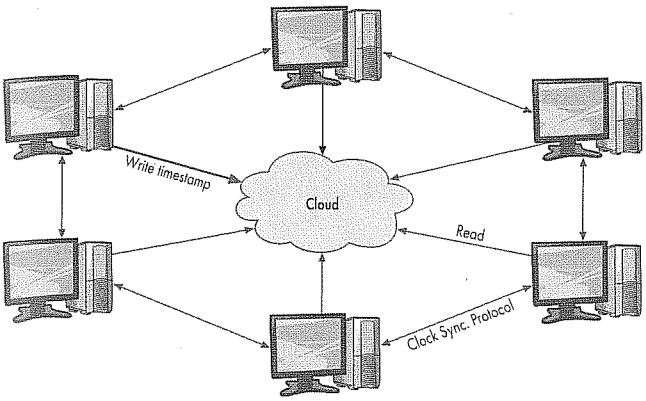
In den letzten Jahren haben Speicherdienste in der Cloud sowohl für Firmen als auch für Privatanwender immer mehr an Bedeutung gewonnen. Beliebige Daten kann man dort unmittelbar („on-demand"), kostengünstig und skalierbar ablegen. Im Gegensatz zu klassischen Datenbanken oder Dateisystemen garantieren diese Speicherdienste üblicherweise jedoch keine strikte, sondern lediglich „letztendliche Konsistenz" („eventual consistency").

Um sie verfügbar zu halten und als Schutz vor Ausfällen speichert der Dienstanbieter Daten mehrfach auf verschiedenen Servern. Diese Replikate sind nicht immer konsistent: ein Schreibzugriff aktualisiert in der Regel nur einen Teil von ihnen, bevor die Anfrage terminiert. Anschließend propagiert das System die Änderungen im Hintergrund asynchron auf die übrigen Server. Dadurch sind, eine Phase ohne Schreibzugriffe vorausgesetzt, nur „letztendlich" alle Kopien nach einiger Zeit wieder konsistent. In diesem Zusammenhang bedeutet Konsistenz nicht die von der Datenbank gewohnte ACID-Eigenschaft. Vielmehr ist gemeint, dass alle Kopien eines Datensatzes identisch sind – dies umfasst insbesondere, dass das Speichersystem für einzelne Schlüssel konsistente und inkonsistente Daten enthält.

Letztendlichkeit messbar machen

Während sich der Preis eines Cloud-Angebots und Qualitätsmerkmale wie Antwortzeiten und Durchsatz relativ einfach ermitteln und vergleichen lassen, scheint es in Bezug auf ihre Konsistenz zunächst keine Unterschiede zu geben – die meisten Dienste garantieren sie nur letztendlich. Es fehlen jedoch genaue Angaben, wann „letztendlich" ist. Die Bestimmung von „Letztendlichkeit" ist vergleichbar mit der Bitte einer Mutter, ihr Sohn möge sein Zimmer aufräumen, und seiner Zusage, das früher oder später zu tun. Für eine Webanwendung mag eine „Letztendlichkeit" von wenigen Millisekunden tolerierbar sein, eine Inkonsistenz von mehreren Sekunden allerdings nicht. Offensichtlich hängt die Länge des Inkonsistenzfensters (die Zeit zwischen dem Schreiben des ersten und des letzten Replikats) vom Systemdesign ab.

Um die Konsistenz vergleichen zu können, hat die Forschungsgruppe eOrganisation am Karlsruher Institut für Technologie (KIT) ein Verfahren entwickelt, das experimentell für beliebige Cloud-Speicherdienste die Dauer der Inkonsistenzfenster approximiert. Für das Verständnis des Vorgehens muss man sich verdeutlichen, wie ein verteilter Speicherdienst funktioniert. Typischerweise hält er mehrere geografisch verteilte Kopien desselben Datums vor. Erreicht ihn eine Leseanfrage, leitet ein Load Balancer sie an einen oder mehrere Server weiter und wartet auf deren Antworten. Unter ihnen identifiziert der Dienst beispielsweise mit Versionsnummern die neueste Version und gibt sie via Load Balancer zurück. Ähnlich verläuft das Schreiben: Der Load Balancer leitet die Anforderung an alle Server weiter und wartet auf alle oder einen Teil der Antworten, bevor er sie bestätigt. Abhängig von der Anzahl der Replikate insgesamt sowie der Menge an Servern, deren Antwort für eine erfolgreiche Lese- oder Schreibanfrage nötig sind, garantiert dieses System unterschiedliche Konsistenzgrade: von strikt (alle Replikate müssen iden-



Im Test schreibt ein Rechner regelmäßig mit Zeitstempel versehene Daten in den Cloud-Speicher. Mehrere Reader lesen sie kontinuierlich, und aus den Unterschieden zwischen Schreib- und Lesezeitpunkt lässt sich das Inkonsistenzfenster ermitteln (Abb. 1).

Beispiel-Messwerte für ein Datum

Zeit	Writer	Reader 1	Reader 2
0	(A,0)	–	–
1		(A,0)	(A,0)
2		(A,0)	(A,0)
3		(A,0)	(A,0)
4		(A,0)	(A,0)
5	B(5)	(A,0)	(B,5)
6		(A,0)	(B,5)
7		(B,5)	(B,5)
8		(A,0) <– 3	(B,5)
9		(B,5)	(B,5)
10	(C,10)	(B,5)	(A,0) <– 5
11		(C,10)	(C,10)
12		(B,5) <– 2	(C,10)
13		(C,10)	(C,10)
14		(C,10)	(B,5) <– 4

tisch sein) bis zu diversen Abstufungen von „Eventual Consistency“ (eine geringe Anzahl von Replikaten reicht für eine Antwort).

Um die Länge des Inkonsistenzfensters zu bestimmen, könnte man eine Schreibanfrage an das Speichersystem schicken und solange die Werte der einzelnen Replikate beobachten, bis alle den neuen Wert angenommen haben, und anschließend die dafür notwendige Zeit bestimmen. Da die Dienste jedoch keinen direkten Zugriff auf die einzelnen Replikate bieten, muss man einen anderen Weg finden.

Dabei hilft es, das Ziel des Load Balancers zu berücksichtigen: Alle Server sollen eine ähnliche Auslastung haben. Im besten Fall reicht es also bei drei Replikaten auf ebenso vielen Servern drei Leseanfragen gleichzeitig zu schicken. Da aber die Funktionsweise des jeweiligen Load Balancer unbekannt ist, sollte man deutlich mehr gleichzeitige Anfragen starten. Verteilt er etwa im Beispiel jeden Request zufällig, ist erst mit neun Anfragen erstmals die Wahrscheinlichkeit größer als 90 %, alle drei Kopien zu lesen – mit 93 % allerdings nicht besonders hoch.

Für das Abschicken von neun parallelen Anfragen braucht man neun Server, die sogenannten Reader. Ebenso ist ein weiterer Server erforderlich, der die Schreibanfrage losschickt (Writer). Da ein Load Balancer häufig ebenfalls die geografische Nähe zu Replikaten berücksichtigt, ist es sinnvoll, die Reader räumlich mindestens ebenso zu verteilen wie diese. Das maximiert die Wahrscheinlichkeit, dass die Reader alle Replikate erwischen, und gleichzeitig die Genauigkeit der Ergebnisse. Um auf Nummer sicher zu gehen, verwendet man weitere Reader über das Minimum hinaus.

Regelmäßig Daten schreiben und lesen

Zusammengefasst kommt man zu einer Anordnung wie in Abbildung 1: Ein Writer schreibt periodisch eine Kombination aus seinem lokalen Zeitstempel und einer Versionsnummer in die Cloud. Gleichzeitig gibt es einige verteilte Reader, die diese Informationen kontinuierlich auslesen. Setzt man die Informationen aller Reader zusammen, lässt sich das Inkonsistenzfenster pro Testlauf durch das Maximum aller gemessenen Unterschiede zwischen Lese- und Schreibzeitpunkt bestimmen. Die Tabelle „Beispiel-Messwerte für ein Datum“ zeigt, wie dies ablaufen kann. Reader 1 liest 3 Zeiteinheiten nach dem Abschicken von Version B noch einmal Version A, bei Reader 2 trifft nochmals zwei Einheiten später erneut Version A ein. Das Inkonsistenzfenster für Version A ist deshalb 5. So lässt sich nur das für den Nutzer beobachtbare Fenster ermitteln, was in der Regel entscheidend ist. Das tatsächliche Inkonsistenzfenster beim Anbieter kann davon leicht nach oben abweichen, etwa wenn er noch ältere Daten vorhält aber nicht ausliefert.

Eines der bekanntesten Cloud-Computing-Angebote kommt von Amazon. Dazu gehört S3 (Simple Storage Service), ein Key-Value-Store, der einzelne Werte (typischerweise Dateien) speichert und durch eindeutige Schlüssel identifiziert. Es gibt nur Operationen zum Lesen, Schreiben und Löschen der Daten als Ganzes, sodass es beispielsweise nicht möglich ist, lediglich einen Teil einer Datei beziehungsweise eines Datums zu ersetzen. Im Gegenzug verspricht S3 Skalierbarkeit sowie eine hohe Verfügbarkeit und Ausfallsicherheit durch Replikation und geografische Verteilung. Da dieser Dienst einer der bekanntesten ist, der ebenfalls nur letztendliche Konsistenz garantiert, lag es nahe, sein Konsistenzverhalten mit dem beschriebenen Verfahren zu untersuchen.

Wiederholt wurden im August 2011 dafür zunächst 14 Instanzen der Elastic Compute Cloud (EC2) in der Region EU-West (Irland) gestartet: Ein Writer, zwölf Reader und ein zusätzlicher

iX-TRACT

- Cloud-Speicherdienste verteilen Kopien desselben Datums auf verschiedene Server. Dabei werden nicht alle Replikate gleichzeitig geschrieben.
- Dadurch entstehen inkonsistente Daten, da einige Kopien jünger sind als andere. Letztendlich gewährleistet der Speicherdienst jedoch Konsistenz über alle Replikate.
- In Untersuchungen von Amazons S3-Service dauerte es einige Hundert Millisekunden bis zu rund zehn Sekunden, bis Konsistenz erreicht war. Insbesondere tagsüber folgten die Inkonsistenzfenster einem charakteristischen Sägezahnmuster.

Server, der die Messdaten aggregiert. Während der Writer alle zehn Sekunden seine Kombination aus Timestamp und Versionsnummer an S3 schickte, lasen die Reader alle zehn Millisekunden – was die Genauigkeit der Ergebnisse limitiert. Intuitiv würde man ungefähr normalverteilte Konsistenzfenster erwarten, bei denen es abhängig von der Systemlast unterschiedlich lange dauert, die Updates zu propagieren. Überraschenderweise war dies bei S3 aber ganz und gar nicht der Fall.

Inkonsistenz sägend oder niedrig

Grob lassen sich die beobachteten Inkonsistenzwerte in zwei alternierende Phasen einteilen: Saw und Low. Diese Namen stehen für die Muster der Inkonsistenzfenster. Während die Low-Phase niedrige, zufällig gestreute Werte im Bereich weniger Hundert Millisekunden zeigt, finden sich in der Saw-Phase „sägende Konsistenzen“: Der erste Testlauf liefert ein Inkonsistenzfenster zwischen rund null und zwei Sekunden, anschließend steigt dieser Wert für sieben oder acht Tests linear an, bis er ein Maximum zwischen zehn und zwölf Sekunden erreicht. Direkt danach beginnt dieses Muster von Neuem, sodass im nächsten Test das Inkonsistenzfenster wieder minimal ist. Dieses Muster wiederholt sich alle 100 Sekunden. Abbildung 2 zeigt einen kurzen Ausschnitt aus einer Saw-Phase.

Diese Saw-Phasen starten werktags gegen acht Uhr morgens und dauern in der Regel bis abends 21 Uhr, freitags enden sie schon früher. Des Weiteren gibt es kurze Saw-Abschnitte verteilt über den Samstagvormittag und den Sonntag. In der restlichen Zeit regiert die Low-Phase. Es lässt sich also ein klarer Zusammenhang zwischen Tageszeit und Inkonsistenzfenster herstellen. Möglicherweise gibt es große Lastunterschiede zwischen den Hauptgeschäftszeiten und außerhalb, die massiv die Qualität des Dienstes beeinträchtigen. Abbildung 3 zeigt hierzu den Verlauf eines einwöchigen Tests beginnend Montagmorgen um 10:30 Uhr.

Batch-Prozesse dürften als Erklärung für das Verhalten nicht infrage kommen, da sie vermutlich eher in der lastarmen Nacht laufen würden, wohingegen schlechte Konsistenzwerte nur tagsüber zu beobachten sind. Eine andere Variante wäre, dass Amazon wegen der höheren Last während des Tages andere Replikationsprotokolle nutzt oder wegen der Vielzahl an Anfragen Filter gegen DDOS-Attacken greifen. Letztlich könnte nur die Firma selbst Licht ins Dunkel bringen.

Unterschiedliche Inkonsistenz im Eimer

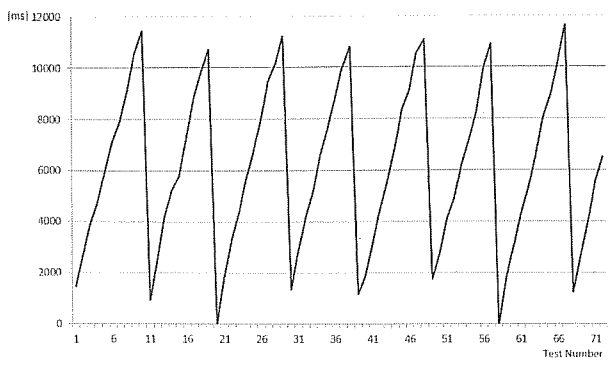
Noch interessantere Muster ergeben sich bei genauerer Berücksichtigung der S3-Struktur: Der Dienst speichert Daten in sogenannten Buckets („Eimer“), die einem Ordner im Dateisystem ähneln. Ein Bucket-Name muss über alle Benutzer hinweg eindeutig sein. Ein entscheidender Unterschied zum Dateisystem ist jedoch, dass sich Buckets nicht verschachteln lassen. Zwei weitere Experimente sollten helfen, den Einfluss der Bucket-Struktur auf die Ergebnisse zu erkennen. In beiden Fällen wurden zwei Dateien gleichzeitig beobachtet: Einmal lagen sie im selben, das zweite Mal in unterschiedlichen Buckets. Derselbe Writer schickte die beiden Dateien jeweils unmittelbar hintereinander an S3, um eine zeitliche Synchronisation beider Writes zu erreichen. Andernfalls hätte die Gefahr bestanden, dass diese immer weiter auseinanderlaufen.

Die Ergebnisse zeigten wieder die Sägezahnmuster in den Saw-Phasen und Zufallswerte in den Low-Phasen. Diesmal allerdings leicht abgewandelt: Während die Daten für die zuerst geschriebene Datei in beiden Fällen aussahen wie vorher, unterschied sich das Ergebnis für die zweite stark davon. In Szenario 1 (beide Dateien im selben Bucket) blieb die zweite Datei konstant in der Low-Phase. In Szenario 2 war das ebenso der Fall, allerdings traten dort im Abstand von etwa sieben Tests einzelne Spitzen auf, bei denen das Inkonsistenzfenster plötzlich zwischen zwei und zehn Sekunden dauerte. Dies korrelierte allerdings nicht erkennbar mit den Sägezahnmustern oder den Low-Phasen der ersten Datei.

Dem von den ersten Tests nahelegten Zusammenhang zwischen Uhrzeit und Konsistenzverhalten widersprechen die Ergebnisse aus den Zwei-Bucket-Experimenten, in denen die jeweils zweite Datei ein anderes Verhalten an den Tag legt. Eine mögliche Erklärung dafür wäre, dass S3 Mechanismen auf Bucket-Ebene verwendet. Beispielsweise könnte die Replikat-Synchronisation unter hoher Systemlast (etwa werktags tagsüber) einerseits in periodischen Abständen erfolgen und andererseits durch eine bestimmte Zahl an Writes erzwungen werden. Wiederum gilt, dass letztlich nur Amazon das beobachtete Verhalten erklären kann.

Amazon veränderte die Technik

Auf Anfrage wollte das Unternehmen keine Informationen bezüglich der Ursachen preisgeben. Ausgehend von den Ergebnissen der KIT-Untersuchungen muss es aber Veränderungen am System vorgenommen haben, denn wenige Wochen nach Kontaktaufnahme mit Amazon zeigte S3 plötzlich leicht veränderte Konsistenzcharakteristika. Während der prinzipielle Unterschied zwischen Saw- und Low-Phasen gleich blieb, veränderten sich beide Abschnitte leicht. In Saw-Phasen sind die Sägezahnmuster weniger deutlich zu erkennen. Sie tauchen immer wieder für etwa zehn Wellenlängen klar auf und gehen danach für kurze Zeit in „zufälligere“ Muster über, bevor sich das Ganze wiederholt. In den Low-Phasen hingegen finden sich jetzt neben der üblichen Zufallsstreuung Perioden von etwa einer Viertelstunde Länge mit einer Vielzahl von Spitzenwerten, bei denen das Inkonsistenzfenster für einen Testlauf schlagartig auf etwa zwei Sekunden springt.

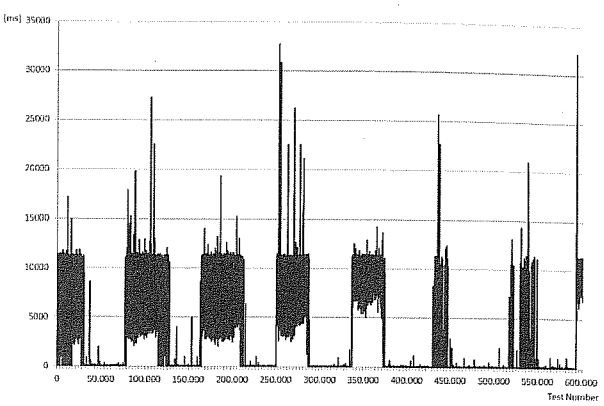


Während der Saw-Phase wird das Inkonsistenzfenster erst immer größer, bis es nach einigen Tests abrupt wieder bei seinem Minimum beginnt (Abb. 2).

Neben der Länge von Inkonsistenzfenstern ist eine andere Konsistenzdimension von Interesse: die monotone Lesekonsistenz („monotonic read consistency"). Sie garantiert, dass ein Lesezugriff keine ältere Version desselben Datums liefert als ein früheres Lesen. Für viele Anwendungsfälle reicht das als Konsistenzgarantie aus. Um bestimmen zu können, wie oft S3 diese Bedingung verletzt, nutzt das Benchmarking-Werkzeug intensives Logging, bei dem es das Ergebnis jedes Lese- und Schreibzugriffs sowie den jeweiligen Zeitpunkt festhält. Dadurch lässt sich für alle gelesenen Werte prüfen, wie oft der Dienst danach einen veralteten liefert. Das Verhältnis der Summe dieser Zahlen zur Gesamtzahl der Lesezugriffe beschreibt die Wahrscheinlichkeit, mit der ein Speicherdienst ältere Daten zurückliefert als bereits gelesen wurden. Während das Inkonsistenzfenster durch das Update etwas wuchs, verbesserte sich die monotone Lesekonsistenz massiv. Vorher bestand eine etwa zwölfprozentige Chance, eine veraltete Version zu lesen. Nach dem Update war dieser Wert auf etwa fünf Prozent gefallen. Diesen Messungen lagen mehr als 353 Millionen Lesezugriffe (vor dem Update) beziehungsweise mehr als 94 Millionen (nach dem Update) zugrunde.

Neben der monotonen Lesekonsistenz und der Länge des Inkonsistenzfensters kann man zwei weitere Konsistenz-Eigenschaften untersuchen: Monotone Schreibkonsistenz („monotonic write consistency") und Schreib-Lese-Konsistenz („read your writes consistency"). Beide erleichtern wesentlich die Programmierung einer Anwendung, die auf dem Speichersystem aufbaut. Monotone Schreibkonsistenz bedeutet, dass das Speichersystem Schreibanforderungen in der Reihenfolge umsetzt, in der der Client sie abschickte. Diese Anforderung ist beispielsweise für Börsenkurse wichtig: Sonst wäre nicht sichergestellt, dass der Handel am Morgen mit dem Schlusskurs des Vortags startet. Schreib-Lese-Konsistenz hingegen bedeutet, dass ein Client immer eine Version lesen kann, die nicht älter ist als sein letzter Schreibzugriff.

Monotone Schreibkonsistenz lässt sich bestimmen, indem ein Client unmittelbar hintereinander zwei Schreibanfragen sendet. Wiederholtes Lesen sollte immer den Wert der zweiten zurückliefern, sonst ist die monotone Schreibkonsistenz verletzt. Ein Nachteil ist hier allerdings zu erwarten: Selbst wenn viele Tests immer den korrekten Wert zurückliefern, garantiert dies nicht, dass immer monotone Schreibkonsistenz gilt, da sie insbesondere bei Fehlern im System gefährdet ist. Angenommen, der erste Request geht an Server A und der zweite an Server B. Fällt anschließend Server A aus und ist erst wieder verfügbar, nachdem B seine Daten weitergegeben hat, könnte die monotone Schreibkonsistenz verletzt werden, da A seinen älteren Wert weitergibt. Mit Mechanismen wie Vector Clocks lässt sich dies zwar leicht verhindern. Aber da man bei einem Cloud-Dienst nicht weiß, ob und welche Fehler auftreten, kann ein Benchmark nicht mit Sicherheit auf diese Fehler prüfen.



Eine einwöchige Aufzeichnung der Inkonsistenzfenster zeigt einen deutlichen Einfluss der Hauptgeschäftszeiten (Abb. 3).

Für Schreib-Lese-Konsistenz besteht dasselbe Dilemma: Nicht abgefangene Fehler können die Korrektheit der Beobachtungen gefährden. Bei der Messung selbst schickt ein einzelner Client einen Schreibrequest an das Speichersystem, gefolgt von einer Vielzahl an Leserequests. Sie müssen für die Gewährleistung der Schreib-Lese-Konsistenz jedes Mal den letzten geschriebenen Wert liefern. Wiederholt man dies oft genug, ist die Wahrscheinlichkeit für ein korrektes Ergebnis hinreichend groß. Diese Bedingung ist bei S3 sowohl für Schreib-Lese- als auch für monotone Schreib-Konsistenz erfüllt.

Fazit

Cloud-Speicherdienste wie S3 bieten kostengünstig hohe Skalierbarkeit und Verfügbarkeit bei geringen Latenzen. Dafür garantieren sie üblicherweise nur reduzierte Konsistenz. Bei Amazons S3 müssen Entwickler also berücksichtigen, dass es manchmal recht lange dauern kann, bis alle Replikate in einem konsistenten Zustand sind. Dies sollte die darüberliegenden Anwendungsschicht beachten, indem sie Abweichungen toleriert. Caching kann hierbei helfen.

Gleichzeitig kann ein Entwickler mit hoher Wahrscheinlichkeit davon ausgehen, dass eine Kopie nicht länger als zwölf Sekunden hinterherhinkt, und das nur während des Tages (Beobachtungen in Europa im Herbst 2011). Nachts sind wesentlich bessere Ergebnisse möglich. Kombiniert man Metainformationen mit den eigentlichen Daten, etwa eine Versionsnummer, kann eine Anwendung selbst auf Verletzungen der monotonen Lese-/Schreibkonsistenz oder der Schreib-Lese-Konsistenz reagieren, etwa durch erneutes Lesen.

Bei der Entscheidung für oder gegen ein bestimmtes Cloud-Speichersystem sollte man sich nicht nur auf funktionale Eigenschaften wie Anfragemächtigkeit, Antwortzeitverhalten und Angaben in den Service Level Agreements (SLA) beschränken, sondern auch andere nichttriviale Qualitätsmerkmale hinterfragen. „Versteckte" Eigenschaften wie Datenkonsistenz sind oft von entscheidender Bedeutung für die Qualität einer Anwendung. Insbesondere kann man immer auf ein strenger konsistentes Speichersystem wechseln – umgekehrt ist dies nur bedingt möglich. (ck)

DAVID BERMBACH

ist Diplom-Wirtschaftsingenieur und promoviert im Bereich Cloud Computing in der Forschungsgruppe eOrganisation am Institut AIFB des Karlsruher Instituts für Technologie.

STEFAN TAI

ist Professor am Karlsruher Institut für Technologie sowie Direktor am FZI Forschungszentrum Informatik in Karlsruhe und in Berlin. Seine Forschungsinteressen liegen in der Gestaltung und Bewertung verteilter Systeme im Cloud Computing.