JAVA Mag inkl. CD

Geo-Tutorial: Laufen im Kreis »37

S&S

Deutschland €8,50 Österreich €9,80 Schweiz sFr 16,80 5.2010

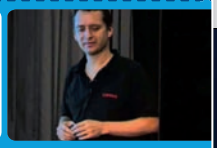
Javamagazin

Java • Architekturen • SOA • Agile

www.javamagazin.de

CD-INHALT

jax TV



Neue Sprachen für die JVM

Video von der W-JAX 2009 in voller Länge

WEITERE INHALTE

Gradle 0.8

Maven 2.2.1

OpenLayers 2.8

Moonlight

Ant 1.8

EclEmma

Apache UIMA 2.3

Apache PDFBox 1.0

Alle CD-Infos ab Seite 3

Cloud Computing
Clouds jenseits des Standards mit BASE »76

Employer Branding
Der Kampf um die besten Fachkräfte »90

jax 2010
Alle Infos im Heft

Build-Tools

★ Maven, Ant und Gradle im Vergleich »60

★ „Maven 3 wird das beste Maven"
Interview mit Jason van Zyl »70

Silverlight für Java

Geht das? »45

Twitter mit Lift

Marke Eigenbau »52

Google Maps und Android

Standortbestimmung »101

4 194586 708507 05

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

Jumping Jack **Flash**

Cloud Computing erobert längst die Herzen und Budgets von Managern und IT-Bereichen. Kommt da wirklich etwas Neues, oder etikettieren wir nur wieder einmal Produkte und Initiativen um? Ist Cloud Computing mehr als nur die Verlängerung der Innovationen rund um Virtualisierung, Standardisierung und Automatisierung? Wir lernen in diesem Artikel die Architekturklassen ACID und BASE zu unterscheiden, warum und wie uns das CAP-Theorem als Architekten eindeutige Entscheidungen abverlangt und schauen in die Wiege des Cloud Computings sowie hinter die Kulissen großer Websites wie Amazon und Google.

von Lothar Wieske

Mittwoch, der 29. Juli 2000 – welche Erinnerungen verknüpfen Sie mit diesem Tag? Vielleicht war es ein schöner Sommertag, den Sie mit Partner(in), Familie, Freunden verbracht haben, vielleicht war es aber eher düster und gewittrig oder es hat wie aus Eimern geschüttet …

Der 29. Juli 2000 war jedenfalls ein geschichtsträchtiger Tag für das Web. Beim ACM Symposium on the Principles of Distributed Computing (PODC) hat Eric Brewer an diesem Tag eine Keynote mit dem Titel „Towards Robust Distributed Systems" gehalten [1]. In diesem Vortrag hat er die Empfehlung ausgesprochen, im Umfeld zunehmend verteilter Systeme und webbasierter Anwendungen bei den Bemühungen um Datenkonsistenz eine gewisse Umsicht und Vorsicht walten zu lassen. Denn bei verteilten Systemen stehen Availability und Consistency der Daten in Konkurrenz.

In seinem Vortrag führte Eric Brewer die Unterscheidung zwischen ACID und BASE ein. Dabei steht ACID für Atomicity, Consistency, Isolation und Durability – diese Abkürzung ist im Umfeld von Datenbanken und Transaktionen bestens eingeführt. Die Abkürzung BASE bedeutet Basically Available, Soft-state und Eventually consistent. In der Gegenüberstellung von Säuren und Basen verbirgt sich gleichzeitig ein eleganter Hinweis auf die Verwendung der Abkürzungen. Dahinter stehen nicht scharf abgegrenzte oder auch nur abgrenzbare

Artikelserie

Teil 1: Amazon Web Services: Flaggschiff des Cloud Computings

Teil 2: Elastic Compute Cloud: Darf es noch ein Server mehr sein?

Teil 3: BASE: Internetanwendungen jenseits der Standards

Anwendungsklassen – vielmehr umreißen beide Konzepte ein Kontinuum von Abwägungen und Entscheidungen. In diesem Kontinuum muss der Architekt den pH-Wert seiner „Lösung" zwischen Availability (BASE) und Consistency (ACID) einpegeln.

Eric Brewer fasste in seinem Vortrag gemeinsame Erkenntnisse und Erfahrungen aus der Arbeit einer Reihe von Menschen der akademischen und industriellen Welt zusammen. Zu jener Zeit war er zum einen Professor an der UC Berkeley und zum anderen Gründer von Inktomi Corporation. Inktomi hatte als Ausgründung der UC Berkeley damals etwa 700 Mitarbeiter und baute Search Engines und Web Caches. Im Dezember 2002 wurde Inktomi von Yahoo! gekauft.

Auf der eigentlichen Schlüsselfolie des Vortrags schaltete Eric Brewer jedoch von zwei auf drei um. Denn in Gänze geht es um drei systemische Merkmale: Availability, Consistency und Partition Tolerance. Und kurz und trocken hieß es dann: „You can have at most two of these properties for any shared-data system." Diese Beziehung ist im Jahr 2000 als Brewer-Hypothese (Brewer Conjecture) in die Annalen eingegangen. Zwei Jahre später haben dann Seth Gilbert und Nancy Lynch vom MIT die Hypothese formal bewiesen [2], und damit entstand eigentlich erst im Jahr 2002 das Brewer-Theorem oder auch CAP-Theorem, weil Consistency, Availability und Partition Tolerance in Beziehung gesetzt werden.

Heute stehen Websites wie Amazon und Google genau in der Nachfolge der grundsätzlichen Abwägung zwischen Availability und Consistency. Deswegen soll es in diesem Artikel um einige der Phänomene im Umfeld des Brewer-Theorems und im Vorfeld des Cloud Computings gehen.

Wirtschaftliche Bedeutung des CAP-Theorems

Die Aberdeen Group hat Ende 2008 eine interessante Untersuchung mit dem vielsagenden Titel „Customers are Won or Lost in One Second" veröffentlicht [4]. Darin wurden 160 Organisationen hinsichtlich ihrer Best Practices bei der Optimierung der Antwortzeiten ihrer unternehmenskritischen Webanwendungen befragt. Die Kernaussagen dieser Untersuchung waren:

- *… business performance (as measured by customer satisfaction, conversions, etc.) begins to suffer at 5.1 seconds of delay in response times of web applications …*
- *… only one second of additional delay in the response times of web application could significantly impact some of the top business goals …*
- *one second of delay in reponse times of web applications can impact customer satisfaction by up to 16 %*

Mit ihren Einsichten und Zahlen hat die Aberdeen Group einen wichtigen Impuls für die gemeinschaftliche Arbeit von Businessanalysten und technischen Architekten gegeben. Gemeinsam sollten beide Funktionsträger fachliche Mengengerüste und technische Plattformunterstützung mit den Geschäftszielen (bspw. Kundenzufriedenheit) vor dem Hintergrund von Kosten, Umsatz und Gewinn ausbalancieren. Klar wird aus den angesprochenen Zahlen und Effekten, dass die Geschäftsmodelle von Amazon (Internethandel) und Google (Internetwerbung) auf Availability fokussieren, und das hat Auswirkungen in den Infrastrukturen dieser Websites.

Es gibt übrigens für die Auswirkungen verzögerter Antwortzeiten bei Amazon und Google interessante Zahlen [4]. Amazon hat herausgefunden, dass jeweils 100 Millisekunden Verzögerung zu einem einprozentigen Umsatzrückgang führen. Untersuchungen von Google weisen für zusätzliche 500 Millisekunden bei der Generierung von Suchergebnissen einen zwanzigprozentigen Rückgang der Anfragen nach.

Technische Bedeutung des CAP-Theorems

Bisher sind die Konzepte Availability, Consistency und Partition Tolerance eher informell in Erscheinung getreten. Gerade vor dem Hintergrund zitierter Beweisführungen ist das natürlich problematisch, weil das mathematische Resultat auf präzise und formale Definitionen aufbaut. Keine Angst, es wird jetzt nicht unnötig kompliziert und es geht nicht um die Wiedergabe der formalen Beweisführung in ihren Tiefen. Aber es geht um ein Verständnis wichtiger Entwicklungslinien in der konzeptuellen Durchdringung von Konsistenz.

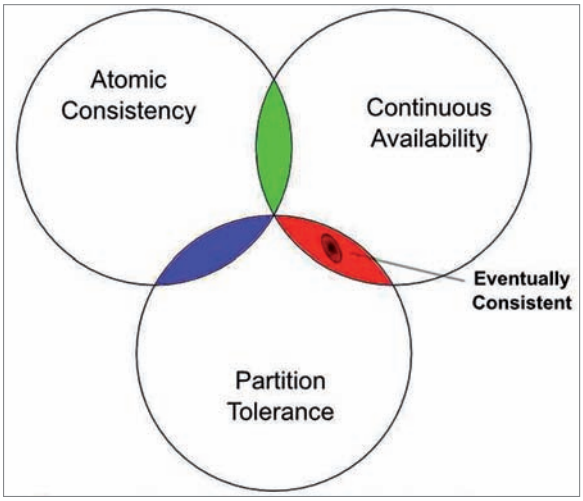


Abb. 1: CAP-Theorem

- Brewer deutet ein Kontinuum Availability/Consistency an und formuliert das CAP-Theorem
- Gilbert und Lynch beweisen das CAP-Theorem und unterscheiden Strong/Weak Consistency

- Vogels (Amazon) verdeutlicht Eventual Consistency als Unterkonzept von Weak Consistency und die Bedeutung für Amazon Dynamo als Storage System in der Umsetzung bei Amazon
- Pritchett (eBay) entwickelt Consistency Patterns zur schrittweisen Optimierung von Availability bei gleichzeitiger Relaxierung von Consistency im Übergang von ACID zu BASE

Gilbert und Lynch sprechen eigentlich nicht von Consistency, sondern von Atomic Consistency. Sie fordern eine totale Ordnung aller Operationen, sodass sie wie eine Perlenkette auf der Zeitschnur aufgefädelt werden können. Und dann soll jede Leseoperation, die nach einer Schreiboperation beginnt, entweder den Wert dieser oder einer späteren Schreiboperation liefern. Für Continuous Availability fordern Gilbert und Lynch, dass jeder Request an das System eine Response liefern muss. Für die Präzisierung von Partition Tolerance wird dem Netzwerk zunächst der Verlust beliebig vieler Nachrichten von einem Knoten zu einem anderen Knoten erlaubt. Partitionierung (in Komponenten) bedeutet dann, dass alle Nachrichten von Knoten in einer Komponente zu Knoten in einer anderen Komponente verloren gehen. Und mit Partition Tolerance macht dieser Verlust von Nachrichten eigentlich nichts weiter aus. Mit der Begriffsbildung Atomic Consistency werden die Gemeinsamkeiten und Unterschiede zu „A" und „C" elegant herausgearbeitet und neu gefasst. Denn Database Atomicity und Database Consistency beziehen sich auf Transaktionen als Operationsfolgen, während im Kontext des CAP-Theorems mit Atomic-Consistency-Eigenschaften einer einzigen Request-/Response-Operation beschrieben werden.

Auf dieser Grundlage beweisen Gilbert und Lynch dann ihre ersten beiden Theoreme: „It is impossible in the ... network model to implement a read/write data object that guarantees the … properties ... Availability (and) Atomic consistency in all fair executions (including those in which messages are lost." Für die Ellipse vor dem Netzwerkmodell steht beim ersten Theorem „asynchronous" und beim zweiten Theorem „partially synchronous". Beim „partially synchronous Network" verfügen die Knoten im Netzwerk noch über Timer. Im Gedankengang des Artikels spielt die Unterscheidung der beiden Netzwerktypen eine Rolle, in diesem Artikel werden wir sie jedoch übergehen. Wesentlich scheint für die hiesigen Belange der Abschnitt mit der Überschrift „Weaker Consistency Conditions". Damit wurde die tiefere Beschäftigung mit Eventual Consistency als praxisfähigem Ausschnitt von Weak Consistency eingeläutet.

Eventual Consistency bei Amazon

Werner Vogels, CTO bei Amazon, vertieft im Artikel „Eventually Consistent" für ACM Queue im Dezember 2008 die Rolle von Amazon Dynamo für die Infrastruktur [5]. Im Untertitel stellt er seinen Ausführungen voran: „Building reliable distributed systems at a worldwide scale demands trade-offs – between consistency and availability". Bei Amazon, Google & Co. gibt es ein Credo: Fehler sind der Normalfall. Dahinter verbirgt sich nicht etwa Defätismus oder Sarkasmus. Vielmehr wird aus dieser Einsicht ein klarer Architektur- und Designauftrag abgeleitet. Heruntergebrochen auf die Themenstellung dieses Artikels heißt das: Network Partitions sind sicher – deswegen kann es Consistency und Availability nicht gleichzeitig geben.

Abhängig davon, was die Serverseite bietet, muss der Entwickler auf Clientseite damit umgehen. Wenn die Serverseite den Schwerpunkt auf Consistency legt, wird es Abstriche im Bereich von Availability geben, und der Entwickler wird Strategien entwickeln müssen für Daten, die zwar geschrieben werden müssen, aber nicht geschrieben werden können. Wenn die Serverseite der Availability den Vorrang gibt, wird es Abstriche im Bereich von Consistency geben und der Entwickler wird Strategien entwickeln müssen, ob und wie er gegebenenfalls auch mit leicht überalterten Daten arbeiten kann.

Für den clientseitigen Umgang müssen zunächst einmal die unterschiedlichen Arten und Eigenschaften von Consistency geordnet und in Beziehung gesetzt werden. Bei Strong Consistency garantiert das System, dass nach einer Schreiboperation alle nachfolgenden Leseoperationen diesen geschriebenen Wert sehen. Bei Weak Consistency garantiert das System nach einer Schreiboperation erst nach dem Eintreten einer Reihe von Bedingungen, dass alle nachfolgenden Leseoperationen diesen geschriebenen Wert sehen. Der Zeitraum zwischen dem Abschluss der Schreiboperation und dem Eintreten aller Bedingungen wird dabei als Inconsistency Window bezeichnet. Eventual Consistency stellt eine Sonderform von Weak Consistency dar in dem Sinne, dass ohne weitere Schreiboperationen letztendlich alle Leseoperationen diesen zuletzt geschriebenen Wert sehen. Wenn keine Fehler eintreten, kann die maximale Größe der Inconsistency Window auch aus unterschiedlichen Kenngrößen des Systems errechnet und angegeben werden.

Wahrscheinlich gestaltet sich Eventual Consistency auch deswegen etwas sperrig, weil es zumindest deutsche Informatiker etwas schaudert, wenn sie einfach ihrem intuitiven Sprachgebrauch folgen. Dann heißt „eventual" eher etwa ig und möglich und nicht letztendlich und schließlich. Das System rauscht also nicht in ein nichtdeterministisches Nirwana von dannen, sondern folgt einer zeitlichen Entwicklung im Sinne von t -> ∞. Zusätzlich benennt Vogels noch einige Variationen und Kombinationen von Eventual Consistency:

- Causal Consistency
- Read-your-writes Consistency
- Session Consistency
- Monotonic read Consistency
- Monotonic write Consistency

Auf der Serverseite haben Vogels und seine Mitarbeiter mit Amazon Dynamo ganze Umsetzungsarbeit geleistet. Amazon Dynamo ist ein Key/Value Storage System, das von vielen internen Diensten genutzt wird. Jeder nutzende Dienst von Dynamo bekommt seine eigene

Instanz, die möglicherweise mehrere Rechenzentren umspannt. Entwurfsziel von Dynamo ist die Möglichkeit der Kontrolle seines Verhaltens entlang der Vorgaben der systemischen Merkmale der Gesamtanwendung. Der Architekt kann sich seine Dynamo-Instanz damit so parametrieren, dass er die Entwicklungsvorgaben hinsichtlich Availability, Consistency und Kosten erfüllt.

Eventual Consistency bei eBay

Dan Pritchett, Technical Fellow bei eBay, entwickelt im Artikel „BASE: An ACID Alternative" für ACM Queue im Juli 2008 einige Consistency Patterns, die über relaxierte Consistency zu optimierter Availability führen [6]. Der Untertitel macht die Motivation klar: „In partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability".

Es gibt zwei grundsätzliche Strategien für das Skalieren von Datenbanken: vertikale und horizontale Skalierung. Vertikale Skalierung hebt die Datenbank auf leistungsfähigere Hardware. Horizontale Skalierung erlaubt mit Database Normalization (Zerlegung in unterschiedliche Spalten) und Database Sharding (Zerlegung in unterschiedliche Zeilen) zwei Taktiken, die miteinander kombiniert werden können. Offensichtlich erfordert eine Strategie der horizontalen Skalierung grundsätzlich die Eigenschaft *Partition Tolerance*. Nach Brewers Theorem müssen deswegen für die Eigenschaften *Consistency* und *Availability* Abwägungen stattfinden und Entscheidungen getroffen werden.

Pritchett illustriert einige Consistency-Überlegungen an einem stark vereinfachten Schema mit zwei Tabellen für ein Auktionssystem. Die Tabelle *User* enthält grundlegende Benutzerinformationen und jeweilige Summenbeträge für Kauf und Verkauf. Die Tabelle *Transaction* enthält für jede Transaktion ihren Einzelbetrag und setzt die IDs für Käufer und Verkäufer in Beziehung. Mit jedem Kauf/Verkauf wird eine Zeile in die Tabelle *Transaction* eingefügt und in der Tabelle *User* werden die entsprechenden Summen aktualisiert:

```
Begin transaction
 Insert into transaction(xid, seller_id, buyer_id,
                                        amount);
 Update user set amt_sold=  ... where id=$seller_id;
 Update user set amt_bought= ... where id=$buyer_id;
End transaction
```

Möglicherweise kann das Consistency Constraint relaxiert werden, dass beide Tabellen *User* und *Transaction* das finanzielle Ergebnis sofort und übereinstimmend wiedergeben (müssen). Wenn zugelassen wird, dass die *User*-Tabellen mit ihren Summenbeträgen erst in einem zweiten Schritt verzögert aktualisiert werden, dann wird die Consistency der Tabellen *User* und *Transaction* kurzzeitig geopfert und verzögert wiederhergestellt. So entsteht folgende gesplittete Transaktion:

```
Begin transaction
 Insert into transaction(id, seller_id, buyer_id,
                                        amount);
End transaction

Begin transaction
 Update user set amt_sold=  ... where id=$seller_id;
 Update user set amt_bought= ... where id=$buyer_id;
End transaction
```

Nun sind die Updates und damit auch die Consistency für die Tabellen *User* und *Transaction* entkoppelt. Die weiteren Transformationen sind im Artikel detaillierter beschrieben, hier soll eine Skizze der zugrunde liegenden Ideen genügen. Im nächsten Schritt kommt Persistent Messaging zum Einsatz, um die Updates der Tabellen *User* und *Transaction* auch transaktionell abzusichern. Die bisherige Lösung ist bislang nur dann fachlich korrekt, wenn die Summenbeträge letztlich nur als Schätzungen verstanden werden, die auch die eine oder andere Transaktion auslassen.

Eine technisch korrekte Umsetzung genauer – und nicht nur geschätzter – Summenbeträge könnte von einer eigenständigen Komponente für Persistent Messaging ausgehen. Dann käme jedoch über 2PC (Two Phase Commit) eine Kopplung von Database und Messaging zustande, die die Effekte der Entkopplung der Updates in den Tabellen *User* und *Transaction* kannibalisieren würde.

Die Verfügbarkeit errechnet sich aus dem Produkt der Verfügbarkeiten von Komponenten. Eine Transaktion, die zwei Datenbanken in einem 2PC einbindet, hat bei einer Verfügbarkeit der Einzeldatenbanken von 99,9 % eine Verfügbarkeit von nur noch 99,8 %. Die reduzierte Verfügbarkeit von 0,1 % bedeutet in der Umrechnung auf den Monat 43 Minuten weniger. Also liegen die Nachrichten genau wie die Tabellen *User* und *Transaction* in derselben Datenbank, um ohne 2PC auskommen zu können und damit keine Auswirkungen hinsichtlich Availability zu haben.

Nun gibt es zwei Verarbeitungsprozesse zum Einstellen einer Transaktion in die Tabelle *Transaction* und zum Nachziehen der Summenbeträge in der Tabelle *User*. Der Prozess zum Nachziehen der Summenbeträge liest Nachrichten aus und verändert Werte für die Summenbeträge. Eigentlich läuft das wieder auf eine 2PC-Situation hinaus. Für die Lösung dieser Problematik wird das Konzept einer idempotenten Operation benötigt. Diese kann einmal oder mehrere Male mit gleichem Resultat angewendet werden. Eine idempotente Operation erlaubt deswegen auch partielle Fehler. Denn wenn sie wiederholt ausgeführt wird, ändert das den Endzustand des Systems nicht in unzulässiger Weise. Nun sind aber Updates im Regelfall nicht idempotent.

Im Beispiel der Auktion muss also eine Buchführung (Tabelle) her, die genau mitschreibt, welche Updates bereits erfolgreich durchgeführt worden sind und welche noch ausstehen. Wenn diese Tabelle ein erfolgreiches Update verzeichnet, wird die zugehörige Nachricht als Arbeitsauftrag gelöscht. Dieses Vorgehen ist tolerant gegenüber partiellen Fehlern, kommt aber ohne 2PC-Mechanismen und damit verschlechterte Availability aus. Im Artikel von Pritchett werden weitere Ideen zur Entkoppelung von Operationen (Ordnung von Nachrichten und Transaktionen) vorgestellt, die im Zusammenspiel in einer optimierten Availability auf Kosten einer relaxierten Consistency resultieren. BASE mit dem Konzept der Eventual Consisten-

cy bietet hierfür einen leistungsfähigen Arbeits- und Denkansatz.

Bedeutung in der und für die Praxis

Bei Amazon, Google und Yahoo! sowie vielen anderen sind Infrastrukturen entstanden, die die soeben vorgestellten Ideen im Umfeld von BASE und Eventual Consistency aufnehmen, weiterentwickeln und umsetzen. Die Systeme Amazon Dynamo, Google File System, Google Bigtable und Yahoo! PNUTS sind in wissenschaftlichen Veröffentlichungen vorgestellt worden. Und diese Veröffentlichungen haben auch zu Nachbauten in der Open-Source-Welt geführt. Es gibt also unterschiedliche Zugänge für eine praktische Vertiefung dieser Themen und in den nächsten Absätzen sollen die prominenten Vertreter im Sinne einer ersten Orientierung kurz vorgestellt werden. Open-Source-Systeme für eigene Experimente sind beispielsweise Cassandra, Dynomite, HBase, Hypertable, Voldemort.

Amazon Dynamo

Vielfach zitiert sind die Einsatzbedingungen von Apache Dynamo: „... even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. …". Apache Dynamo ist ein „highly available key-value storage system". Dynamo besteht aus einem Netz von vollständig gleichberechtigten Rechnern. Es gibt keine zentrale Steuerung und jeder Knoten kann jede Aufgabe wahrnehmen. Damit skaliert die Architektur von Amazon Dynamo in einfachster Weise durch das Hinzufügen weiterer Knoten. Um die gewünschten Eigenschaften zu erreichen, werden eine Reihe von bekannten Verfahren genutzt. Consistent Hashing, Gossip, Hinted Handoff, Sloppy Quorum und Vector Clocks sind relevante Stichworte, die zeigen, dass für ein vertieftes Verständnis von Amazon Dynamo einige Vorarbeiten erforderlich sind [7].

Google File System und Bigtable

Google hat mit dem Google File System (GFS) ein verteiltes Dateisystem gebaut, das für große Dateien mit mehreren GB optimiert ist. Inhalte werden nur selten gelöscht oder überschrieben – sie werden aber oft mit hohen Durchsatzanforderungen gelesen – außerdem werden oft Inhalte angehängt. Ein GFS Cluster besteht aus einem Master und vielen Chunk-Servern. Mehrere hundert oder tausend Chunk-Server speichern Chunks als 64 MB große Ausschnitte von Dateien. Der Master erhält alle Anfragen für eine Datei und liefert als Antwort die dazugehörigen Chunk-Server. Der Client holt dann direkt von den Chunk-Servern die benötigten Chunks für seinen Dateizugriff. Google Bigtable ist ein performanter skalierbarer Datastore auf der Grundlage einer „sparse distributed multi-dimensional sorted map". Er baut u. a. auf dem Google File System auf und liegt u. a. der Google App Engine zugrunde [8], [9].

Yahoo! PNUTS

PNUTS (Platform for Nimble Universal Table Storage) ist *„a massively parallel and geographically distributed database system" für die Webanwendungen bei Yahoo!*. PNUTS wurde für den Betrieb in mehreren und über mehrere Rechenzentren entworfen und bietet dafür „per-record timeline consistency", d. h. alle Repliken eines Records werden in derselben Reihenfolge aktualisiert. Records sind versioniert – ein Client kann bei Leseoperationen nach der neuesten Version fragen oder auch einen leicht veralteten Record zulassen. Damit lässt sich clientseitig die benötigte Consistency wählen und einstellen [10].

Zusammenfassung

CAP-Theorem und Cloud Computing – was hat das miteinander zu tun? Nun, Eric Brewer hat in seinem Vortrag eine Tür aufgestoßen und ein Kontinuum aufgespannt. ACID vs. BASE bzw. Consistency vs. Availability. Im CAP-Theorem wurde dieses Kontinuum um eine dritte Dimension erweitert: Partition Tolerance.

Größe, Verteilung und Wachstum gehören zu den zentralen Herausforderung für die Websites und Clouds von Amazon, Google und Co. Größe und Verteilung führen zu einem veränderten Umgang mit Ausfällen und Fehlern – Wachstum erfordert eine in Architektur und Design verankerte Fähigkeit zur Skalierung. Aus den Arbeiten von Brewer, Gilbert und Lynch wird klar, dass die Verfügbarkeits- (Availability) und Verteilungsanforderungen (Partition Tolerance) auch im Cloud Computing zu einem veränderten Umgang mit Consistency führen müssen. „A" und „P" werden nur auf Kosten von „C" möglich. Nun haben Amazon, Google und Co ihre Hausaufgaben gemacht. Mit Amazon Dynamo und Google Bigtable sind Infrastrukturen mit neuartigen Ansätzen für Consistency und Scalability entstanden. Diese Infrastrukturen können und müssen in einfacher Weise durch das Hinzufügen zusätzlicher Compute-/Network-/Storage-Ressourcen wachsen – ihre Architektur liefert die Möglichkeit, und ihre Größe schafft die Notwendigkeit. Und damit kann die eigene Infrastruktur dann auch für die Mitnutzung geöffnet werden. Der Ausbau des eigenen Geschäftsmodells auf der Grundlage neuer Availability- und Scalability-Dimensionen schafft so die Grundlage für den Einstieg in ein weiteres Geschäftsmodell – Cloud Computing. ■

Lothar Wieske ist Enterprise Architect. Als Architekt hat er Anwendungslandschaften entworfen, und als Coach hat er Teams bei Veränderungen begleitet. Seine Themen sind Cloud Computing, Model-driven Development sowie systemisches Coaching und agile Entwicklung.

Links & Literatur

[1] http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[2] http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf

[3] http://www.gomez.com/download/Aberdeen_WebApps.pdf

[4] http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it

[5] http://queue.acm.org/detail.cfm?id=1466448

[6] http://queue.acm.org/detail.

[7] http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

[8] http://labs.google.com/papers/gfs-sosp2003.pdf

[9] http://labs.google.com/papers/bigtable-osdi06.pdf

[10] http://www.brianfrankcooper.net/pubs/pnuts.pdf