Cassandra, ich hör Dich rufen

Cassandra – die NoSQL-Datenbank für große Mengen und schnelle Zugriffe

Lothar Wieske

Cloud Computing stellt höchste Anforderungen an Dateisysteme und Datenbanken. Mit Klassikern wie dem Google Bigtable und Amazon Dynamo haben die Pioniere neue Architekturkonzepte hervorgebracht. Apache Cassandra (Facebook) ist ein Beispiel für eine verteilte, skalierbare, performante und schemafreie Datenbank mit Eventual Consistency. Cassandra hebt ein Bigtable-Datenmodell auf eine Dynamo-Plattform, d. h. Cassandra bietet einen strukturierten Speicher für Schlüssel/Werte-Zuordnungen mit Eventual Consistency. Cassandra ist in Java implementiert und lädt zum Experimentieren ein.

Antwortzeiten, Antwortzeiten, Antwortzeiten ... und immer an die Anwender denken. In einer Untersuchung mit dem Titel *„Customers are Lost or Won in One Second"* hat die Aberdeen Group Best Practices zur Optimierung der Performanz von Webanwendungen in Unternehmen untersucht [Abe08]. Die Teilnehmer der Untersuchung verbanden eindeutige geschäftliche und wirtschaftliche Zielsetzungen mit den technischen Optimierungen: zufriedenere Kunden, produktivere Mitarbeiter, niedrigere Infrastrukturkosten und weniger Störungen bei zentralen Geschäftsprozessen. Und die Schlüsselaussagen der Studie rütteln auf:

- *„... 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 applications could significantly impact some of the top business goals ..."*
- *„... One second delay in response times of Web applications can impact customer satisfaction by up to 16% ..."*

Wer konkrete Zahlen für einzelne Unternehmen sucht, findet sie beispielsweise im Vortrag von Martin Kliehm, Senior Frontend Engineer bei Namics [Kli10]:

- Amazon: 100 ms Verzögerung => -1 % Umsatz
- Bing: 2000 ms Verzögerung => -4,3 % Umsatz
- Google: 400 ms Verzögerung => -0,59 % Suchen pro Anwender
- Yahoo!: 400 ms Verzögerung => 5-9 % weniger Datenverkehr

Seien es Warenumsätze oder Werbeeinnahmen, bei Unternehmen dieser Größenordnung geht es bei solchen Rückgängen um viel Geld und große Verluste. Da muss die Infrastruktur rund um die Uhr laufen und durch hohe Verfügbarkeit und niedrige Verzögerungen glänzen, im direkten Interaktionsverhalten genauso wie bei den Services im Hintergrund. Die virtuellen Warenkörbe der Kunden eines Internethändlers beispielsweise müssen bei jederzeitiger Les- und Schreibbarkeit das Abrauchen eines einzelnen Servers wie auch das Wegbrechen eines ganzen Rechenzentrums überleben.

Genau diese hochverfügbaren Warenkörbe bietet Dynamo bei Amazon. Die großen Suchmaschinen Google und Yahoo! haben mit Bigtable und PNUTS ähnliche Systeme aufgebaut.

CAP-Theorem und Eventual Consistency

Wie schon in [Wies10] ausgeführt, hat Eric Brewer, seinerzeit Professor an der Universität Stanford und Mitgründer der Inktomi Corporation, in seiner Keynote beim ACM Symposium on the Principles of Distributed Computing [Brew00] die Anwendungsklassen

- ACID (Atomicity, Consistency, Isolation, Durability) und
- BASE (Basically Available, Soft State und Eventual Consistency)

als widerstreitende Extreme in einem Kontinuum möglicher Informationsarchitekturen unterschieden. Im Kern seines Vortrags stand eine Vermutung bezüglich Consistency, Availability und Partition Tolerance: *„You can have at most two of three properties in any shared-data system."* Im Jahr 2002 haben Seth Gilbert und Nancy Lynch vom MIT die Vermutung formal bewiesen und sie damit zum Brewer-Theorem oder auch CAP-Theorem erhoben [GiLy02].

Strong Consistency gibt es in zwei Ausbaustufen für die Entitäten im Datenbanksystem. Der einfachere Fall klammert alle Änderungen für eine einzelne Entität, der schwierigere Fall klammert viele Änderungen in einer ganzen Sammlung von Entitäten. Möglicherweise liegen diese Entitäten sogar in mehreren Datenbanken oder Nachrichtensystemen und die müssen dann mit einem Transaktionsmonitor koordiniert werden. Und diese Strong Consistency – eine „Angewohnheit" bei Unternehmensanwendungen – muss es gar nicht immer sein. Eventual Consistency kommt wesentlich entspannter daher, genügt in vielen Fällen vollauf und erschließt ganz neue Bereiche für die Availability. Und beim Cloud Computing spielen Systeme mit Eventual Consistency vom Beginn an eine tragende Rolle [Vog08].

Im Folgenden geht es um Cassandra, eine „Not only SQL"-Datenbank mit Eventual Consistency.

Apache Cassandra

Cassandra ist ein verteiltes Datenbanksystem für viele Daten und große Lasten. Facebook hat es als Speichersystem für die

Suche in den Eingangskörben der Postfächer mit ca. 100 Millionen Benutzern entwickelt und im Juni 2008 in Betrieb genommen. Nach der Freigabe als Open-Source-Projekt im Juli 2008 firmiert Apache Cassandra seit Februar 2010 als Top-Level-Projekt der Apache Software Foundation [Cassandra]. Jonathan Ellis ist Project Lead, war bis vor kurzem Mitarbeiter von Rackspace und hat jetzt seine eigene Firma Riptano gegründet, die Service und Support für Cassandra anbietet.

Cassandra wird heute bei Facebook, Digg, Twitter, Reddit, Rackspace, Cloudkick, Cisco, SimpleGeo, Ooyala, OpenX und anderen bekannten Unternehmen eingesetzt. Das größte bekannte Cassandra-Cluster im Produktionsbetrieb hat mehr als 150 Knoten mit einem Datenvolumen von mehr als 100 Terabyte.

Cassandra-Datenmodell

Das Datenmodell von Cassandra (s. Abb. 1) beruht auf den Konzepten Column, SuperColumn und ColumnFamily. Eine Column ist ein Tripel (name, value, timestamp) mit einem binären Namen, einem binären Wert und einem Zeitstempel.

Eine ColumnFamily bündelt in Rows geordnete Listen von Columns. Der Zugriff auf eine Row erfolgt über den Key und innerhalb der Row auf die Column über deren Name. Eine SuperColumn bündelt ebenso wie eine ColumnFamily Listen von Columns. Die Visualisierungen für eine Column-Family und eine SuperColumn sind also gleich – Austauschen der Bezeichnung ganz oben genügt.

Mit einer SuperColumn erhält eine ColumnFamily noch eine Schachtelungstiefe mehr. Denn eine ColumnFamily bündelt neben Columns auch SuperColumns – nur keine Mischung aus beiden. Abbildung 3 gilt also für eine Standard-ColumnFamily. Abbildung 4 zeigt eine Super ColumnFamily.

Oberhalb der ColumnFamily gibt es nur noch den sogenannten Keyspace – die Keys im Keyspace bestimmen letztlich, auf welchen Knoten im Cluster die zugehörige ColumnFamily gespeichert wird.

Anwendungen optimieren die Ablage und Abfrage von Inhalten in Cassandra durch die Vorgabe einer Sortierung für eine Column in einer SuperColumn oder in einer (Standard) ColumnFamily. Cassandra legt sie so ab und gibt sie auch so zurück.

Die Programmierschnittstelle bietet drei einfache Methoden für das Abfragen, Einfügen und Löschen von Inhalten und ist mit Thrift realisiert, einem Werkzeug von Facebook zur Generierung von Remote Procedure Calls für unterschiedliche Programmiersprachen. Cassandra migriert derzeit von Thrift RPC zu Avro RPC.

Cassandra-Plattform

Die Architektur von Cassandra nutzt einige Schlüsseltechniken für verteilte Systeme. Die Mechanismen für Partitionierung, Replizierung und Caching spielen beim Lesen und Schreiben im Cassandra-Cluster zusammen. Typischerweise stellt ein Client seine Anforderungen von Lese-/Schreiboperationen für einen Schlüssel an einen beliebigen Knoten im Cluster. Dieser Knoten bestimmt dann die Repliken für diesen Schlüssel im Cluster und fragt sie an. Der Storage Proxy behandelt Leseoperationen je nach geforderter Consistency:

- ONE liefert das Ergebnis des ersten antwortenden Knotens.
- QUORUM liefert das aktuellste Ergebnis der Mehrheit der Repliken.

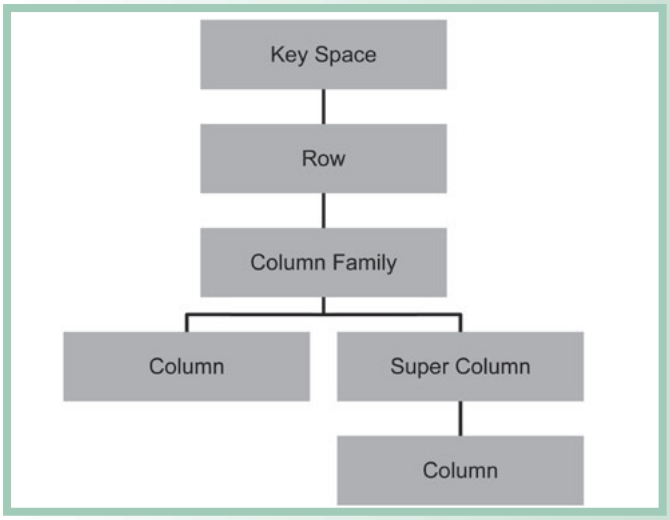

Abb. 1: Cassandra-Datenmodell

Column		
Name	Value	Timestamp
„Vorname“	„Hugo“	127001234

Abb. 2: Cassandra-Column

ColumnFamily			
Key	Columns		
„1“	Name	Value	Timestamp
	„Vorname“	„Hugo“	127001234
	„Nachname“	„Müller“	127001234
„2“	Name	Value	Timestamp
	„Vorname“	„Anna“	127001237
	„Nachname“	„Schmidt“	127001237

Abb. 3: Cassandra-(Standard)-ColumnFamily

SuperColumn			
Key	Columns		
„Post“	Name	Value	Timestamp
	„Subject“	„Hugo“	127001241
	„Body“	„Lorem Ipsum“	127001241
	„Date“	„2010/08/01“	127001241
„Tags“	Name	Value	Timestamp
	„1“	„cloud“	127001247
	„2“	„nosql“	127001247

Abb. 4: Cassandra-(Super)-ColumnFamily

- ALL liefert das aktuellste Ergebnis der Gesamtheit aller Repliken.

Für Schreiboperationen sind die Konsistenzniveaus noch feingranularer:

- ZERO schreibt asynchron im Hintergrund und bietet keine Garantien.
- ANY garantiert, dass mindestens eine Replik geschrieben wurde.
- ONE garantiert, dass mindestens eine Replik mit COMMIT-LOG und MEMTABLE geschrieben wurde.
- QUORUM garantiert, dass die Mehrheit der Knoten geschrieben wurde.
- ALL garantiert, dass die Gesamtheit der Repliken geschrieben wurde.

Cassandra-Partitionierung

Die Fähigkeit zur schrittweisen Skalierung ist ein wesentliches Merkmal von Cassandra. Dazu müssen die Daten dynamisch den vorhandenen Knoten zugeteilt werden (Partitionierung) und Cassandra verwendet dafür eine konsistente Hashfunktion. Dabei wird der Wertebereich der Hashfunktion als Ring organisiert, in dem auf den höchsten Hashwert wieder der niedrigste Hashwert folgt. Jeder Knoten im Cassandra-Cluster bekommt auf diesem Ring einen Hashwert zugeordnet, das sogenannte Token.

ColumnFamily				
Key	SuperColumns			
„Post001“	Key	Columns		
	„Post“	Name	Value	Timestamp
		„Subject“	„Hugo“	127001241
		„Body“	„Lorem Ipsum“	127001241
		„Date“	„2010/08/01“	127001241
	„Tags“	Name	Value	Timestamp
		„1“	„cloud“	127001247
		„2“	„nosql“	127001247
„Post002“	Key	Columns		
	„Post“	Name	Value	Timestamp
		„Subject“	„0.6.3 Released“	127001249
		„Body“	„Lorem Ipsum“	127001249
		„Date“	„2010/06/30“	127001249
	„Tags“	Name	Value	Timestamp
		„1“	„apache“	127001249
		„2“	„cassandra“	127001249
		„3“	„java“	127001249

Abb. 5: Neuer Schlüssel

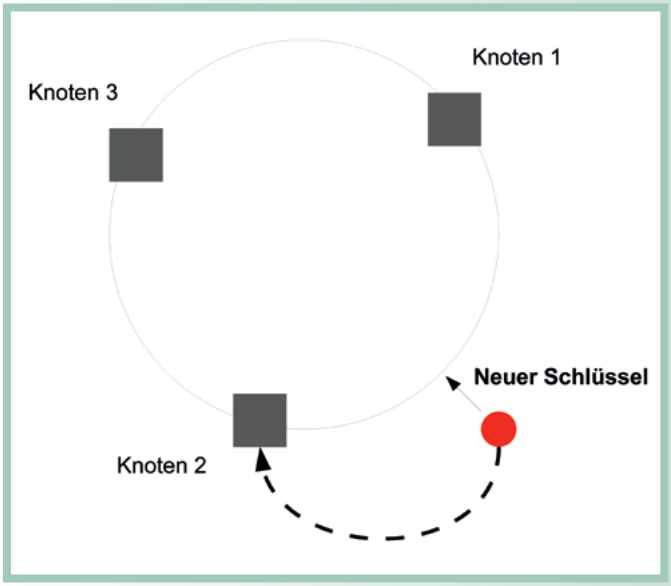

Abb. 6: Neuer Schlüssel

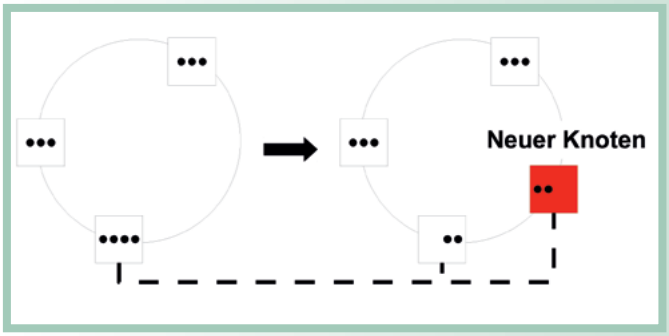

Abb. 7: Neuer Knoten

Für jedes Key/Value-Paar wird für den Key (s. Abb. 5) der Hashwert berechnet und im Ring das im Uhrzeigersinn nächstgelegene Token gesucht und das Key/Value-Paar auf diesem Knoten gespeichert; dieser Knoten ist der Koordinator für das Key/Value-Paar.

Jeder Knoten im Cassandra-Cluster ist also für die Key/Value-Paare zuständig, deren Hashwerte zwischen dem eigenen Token und dem Token des Vorgängers im Ring liegen. Der Eintritt oder Austritt eines Knotens betrifft nur den unmittelbaren Vorgänger und Nachfolger im Ring und bleibt somit auf die Operationen zur Behandlung des Hinzukommens bzw. des Wegfalls eines Tokens auf nur drei Knoten beschränkt.

Dynamo und Cassandra optimieren die konsistente Hashfunktion für ihre Zwecke übrigens etwas unterschiedlich – Cassandra ist eben kein Dynamo-Klon.

Die Zuordnung von Key/Value-Paaren zu Knoten muss keine Hashfunktionen benutzen. Über die Wahl der Partitionierer lässt sich die Zuordnung von Keys zu Token anpassen. Der RandomPartitioner verteilt typischerweise die Keys gut und gleichmäßig, hat aber den Nachteil ineffizienter Bereichsabfragen (engl. range query); der OrderPreservingPartitioner erlaubt effiziente Bereichsabfragen und birgt die Gefahr einer schlechten und ungleichmäßigen Verteilung der Schlüssel.

Cassandra-Replizierung

Jedes Key/Value-Paar wird auf mehreren Knoten repliziert; die Zahl der Repliken im Cassandra-Cluster ist über den Re-

plicationFactor (N) konfigurierbar. Der Koordinator ist für die Replikation auf (N-1) Knoten zuständig. Für die Auswahl von Knoten für die Repliken bietet Cassandra mehrere Optionen an und kann dabei auch die Topologie der Racks* und Rechenzentren einbeziehen (rack.properties, datacenter.properties):

- RackUnaware platziert N-1 Repliken auf den nachfolgenden Knoten im Ring.
- RackAware platziert die zweite Replik in einem anderen Rechenzentrum und die weiteren N-2 Repliken auf anderen Racks im gleichen Rechenzentrum.
- DatacenterShard platziert M Repliken in einem anderen Rechenzentrum und weitere (N-M-1) Repliken auf anderen Racks im gleichen Rechenzentrum.

Eigene Strategien für die Replikation können über die Klasse `org.apache.cassandra.locator.AbstractReplicationStrategy` implementiert werden.

Cassandra-Caching

Cassandra kennt zwei Speicherebenen auf jedem Knoten im Cassandra-Cluster. Die MEMTABLEs befinden sich im Hauptspeicher und COMMITLOG und SSTABLEs (Sorted String Table) liegen im Dateisystem. Das sequenzielle und rollierende COMMITLOG liegt im Dateisystem – am besten auf einer eigenen Platte. Für jede ColumnFamily gibt es eine MEMTABLE, in der Key/Value-Paare sortiert nach Schlüsseln vorgehalten werden. Ebenso gönnt sich jede ColumnFamily eine SSTABLE.

Für jede SSTABLE gibt es drei Dateien im Dateisystem. Eine Datendatei nimmt die Key/Value-Paare sortiert nach Keys auf. Eine Indexdatei liefert die Verweise in die Datendatei als Key/Offset-Paare. Die Filterdatei enthält Buckets für einen Bloom-Filter. Dieser dient der effizienten Prüfung, ob ein Key zur Menge der Keys in der Datendatei gehört.

Dabei werden für alle Keys der Menge mehrere Hashfunktionen berechnet und dann die entsprechenden Buckets für jeden Hashwert markiert. Für einen gesuchten Key werden ebenfalls die Hashwerte berechnet und die Buckets mit dem Bloom-Filter abgeglichen. Wenn alle Buckets markiert sind, gehört der Key wahrscheinlich zur Menge; wenn nur ein Bucket nicht markiert ist, gehört er sicher nicht dazu.

Das Dateisystem eines Knotens im Cassandra-Cluster besitzt also folgenden schematischen Aufbau:

```
./commitlog/CommitLog-ID.log
./data/KEYSPACE/COLUMN_FAMILY-N-Data.db
./data/KEYSPACE/COLUMN_FAMILY-N-Filter.db
./data/KEYSPACE/COLUMN_FAMILY-N-Index.db
...
./system.log
```

Beim Schreiben wird die Änderung zunächst ins COMMITLOG geschrieben und anschließend in der MEMTABLE nachgezogen. Von Zeit zu Zeit wird die MEMTABLE als SSTABLE herausgeschrieben (flush). Ein automatisches Herausschreiben wird angestoßen, wenn einer von drei Schwellwerten überschritten wird: Menge der Daten in der MEMTABLE, Anzahl der Columns in der MEMTABLE und Lebensdauer der MEMTABLE. Für ein manuelles Herausschreiben steht ein Werkzeug zur Verfügung. Nach dem Herausschreiben wird eine SSTABLE nicht mehr geändert. Schreiboperationen benötigen für ihre Änderungen am Ende des COMMITLOG und in der MEMTABLE wenig Zeit und erreichen in einem Cluster mit genügend vielen Knoten zur Lastverteilung schier atemberaubende Durchsatzraten.

Beim Lesen werden die angeforderten Daten aus der MEMTABLE und allen SSTABLEs aufgesammelt. Auch mit Bloom-Filtern zur Optimierung fallen dabei einige Zugriffe im Dateisystem an. Um die Zahl der anzusprechenden SSTABLEs für die Leseoperationen zu minimieren, werden gelegentlich mehrere alte SSTABLEs zu einer neuen SSTABLE zusammengefasst (compact).

Beispiele mit Cassandra

Cassandra ist schnell installiert. Einfach die Binärdistribution entpacken und CommitLogDirectory und DataFileDirectory in der Datei conf/storage-conf.xml geeignet setzen. Cassandra benötigt in aktuellen Versionen Unterstützung für Java 6. Für den Start des Servers reicht ein einfaches

```
bin/cassandra -f
```

Drei virtuelle Maschinen unter VirtualBox oder VMware Player/Workstation oder auch drei Instanzen in der Elastic Compute Cloud (EC2) von Amazon reichen für erste Experimente mit Partitionierung, Replizierung und Caching vollkommen aus.

Beispiel-Partitionierung

Nach dem Start des ersten Servers gibt man mit Kommandozeilenwerkzeug 1000 Datensätze ein.

```
$ bin/cassandra-cli --host XXX.YYY.ZZZ.231 --port 9160
cassandra> set Keyspace1.Standard2['jsmith1000']['first'] = 'John'
cassandra> set Keyspace1.Standard2['jsmith1000']['last'] = 'Smith'
cassandra> set Keyspace1.Standard2['jsmith1000']['age'] = '42'
cassandra> set Keyspace1.Standard2['jsmith1001']['first'] = 'John'
...
```

Sicher strotzen die Datensätze nicht gerade vor Kreativität und Fantasie. Aber sie zeigen das Phänomen und darum geht es ja zum Einstieg.

Der erste Server ist ein sogenannter SEED-Server. Bei seiner Konfiguration bekommt die <seed> den Wert „127.0.0.1“ und <AutoBootstrap> wird auf den Wert „false“ gesetzt. Bei der Konfiguration des zweiten und dritten Servers (das sind sogenannte NON-SEED-Server) bekommt <seed> jeweils die IP-Adresse des ersten Servers und AutoBootstrap wird auf den Wert „true“ gesetzt.

Diese Einstellungen liegen in der Datei conf/storage-conf.xml. Für den ersten Server passen die Grundeinstellungen; erst für den zweiten und dritten Server müssen die Anpassungen erfolgen. Im Oktober 2010 soll übrigens Cassandra 0.7 kommen. Dann wandern die Einstellungen in die Datei conf/cassandra.yaml.

Das Ergebnis zeigt ein weiteres Werkzeug.

```
$ bin/nodetool -host XXX.YYY.ZZZ.233 ring
```

* Der Begriff Rack steht für ein genormtes Rechnergestell für mehrere Rechner im Rechenzentrum. Die Racks für Rechenzentren sind etwa 2 Meter hoch und bieten Platz für 42 Höheneinheiten; eine Höheneinheit ist mit 1,75 Zoll (4,45 cm) spezifiziert. Die Geräte weisen Bauhöhen mit ganzzahligen Vielfachen von einer Höheneinheit auf und haben genormte Breiten von 19 Zoll (48,26 cm). Racks gibt es in Bautiefen von 60, 80, 100 oder 120 cm und Baubreiten von 60, 70 oder 80 cm. Die größeren Baubreiten bieten seitliche Kanäle für Netz, Strom und Luft. Die Racks können mit Seitenwänden und Türen zu Schränken geschlossen werden.

```
Address         Status     Load          Range      Ring
                                         1375...
XXX.YYY.ZZZ.233 Up         495 bytes      521...    |<--|
XXX.YYY.ZZZ.232 Up         495 bytes     1368...    |   |
XXX.YYY.ZZZ.231 Up         495 bytes     1375...    |-->|
```

Beispiel-Caching
Die niedrige Zahl bei Load erklärt sich aus dem Caching, denn hier wird nur Festplattenbelegung angezeigt – die MEMTABLE leert man mit `nodetool ... flush`. Jetzt kann man der Reihe nach den ersten und zweiten Server außer Betrieb nehmen

```
$ bin/cassandra-cli --host XXX.YYY.ZZZ.231 decommission
$ bin/cassandra-cli --host XXX.YYY.ZZZ.232 decommission
```

und schaut sich wieder den Ring an: Die 1000 Datensätze sind vom ersten auf den dritten Server umgezogen.

Beispiel-Replizierung
In der Datei conf/storage-conf.xml kann man dann den ReplicationFactor erhöhen und den Ablauf wiederholen.

Im Kleinen verhilft dieses Vorgehen zum Verständnis der Abläufe bei Partitionierung, Replizierung und Caching und des Betriebs eines verteilten, skalierbaren Datastores.

Zusammenfassung

Apache Cassandra hebt ein Bigtable-Datenmodell [Big06] auf eine Dynamo-Plattform [Dyn06] und kann prominente Einsatzszenarien vorweisen. Der erfolgreiche Einstieg ins Cloud Computing und insbesondere die Arbeit mit oder der Aufbau von geeigneten Speicherkonzepten setzt einerseits eine theoretische Kenntnis der Klassiker wie Bigtable und Dynamo voraus und andererseits die praktische Arbeit mit einem frei verfügbaren System wie Cassandra. Darüber hinaus zeigen die verbauten Entwurfsmuster bei Cassandra, dass elastische und robuste Systeme auf einfachen Prinzipien beruhen und gerade deswegen zu beeindruckender Größe und Form auflaufen. Vielleicht konnte dieser Artikel den Appetit für eigene Vertiefungen anregen. Viel Spaß dabei.

Literatur

[Abe08] B. Simic, Customers Are Won or Lost in One Second, Whitepaper Aberdeen Group, November 2008, `http://www.gomez.com/download/Aberdeen_WebApps.pdf`

[Big06] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, Bigtable: A Distributed Storage System for Structured Data, ODSI 2006, `http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/de//papers/bigtable-osdi06.pdf`

[Brew00] E. A. Brewer, Towards Robust Distributed Systems, PODC Keynote, 19.7.2000, `http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf`

[Cassandra] `http://cassandra.apache.org`

[Dyn06] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's Highly Available Key-Value Store, Proc. of the 21st ACM Symposium on Operating Systems Principles, Oktober 2007, `http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf`

[GiLy02] S. Gilbert, N. Lynch, Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, `http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf`

[Kli10] M. Kliehm, Web Performance Optimierung, Vortrag am 17.5.2010 beim Webmontag in Frankfurt, `http://www.slideshare.net/kliehm/performancewmfra`

[Vog08] W. Vogel, Eventually Consistent – Revisited, `http://www.allthingsdistributed.com/2008/12/eventually_consistent.html`

[Wies10] L. Wieske, Dateisysteme und Datenbanken im Cloud Computing, in: JavaSPEKTRUM, 5/2010

Lothar Wieske ist Unternehmensarchitekt für Innovationsmanagement bei der DB Systel GmbH (ICT Tochter im Deutsche Bahn Konzern). Vorher hat er u. a. im Consulting für IBM, Microsoft und Sun Microsystems gearbeitet und schöpft aus Erfahrungen in Finance, Health und Transport & Logistics. Er beschäftigt sich schwerpunktmäßig mit Cloud Computing und ist Autor eines Fachbuchs.
E-Mail: lothar.wieske@deutschebahn.com