Einsatz von (No)SQL-Datenbanken am Beispiel von Facebook

Problemseminar NoSQL-Datenbanken

Universität Leipzig – Abteilung Datenbanken

- Michael Gassner -

Betreuerin: Anika Groß

Inhaltsverzeichnis

1. Einleitung

Die Menge der weltweit gespeicherten Daten nimmt seit Jahren explosionsartig zu. Unternehmen sammeln und speichern alle erdenklichen Daten über ihre Kunden, Zulieferer und Abläufe. Unzählige Sensoren stecken in nahezu jedem Gerät mit des täglichen Lebens und diese zeichnen Daten auf, speichern und verteilen sie weiter. Die Verbreitung von Video-, Audio- und Bildmedien kombiniert mit immer steigender Qualität, bei gleichzeitiger Verfügbarkeit, ist für ein Datenwachstum mit exponentiellen Ausmaßen verantwortlich. Zusätzlich lassen Menschen, dank Smartphones und sozialen Netzwerken, jeden an allem in ihrem Leben teilhaben. Gleichzeitig hat die Wirtschaft erkannt, dass diese Daten bares Geld sind und ganze Wirtschaftszweige leben von dem Sammeln und Verkaufen dieser Daten. Deswegen werden diese Daten erfasst, nachhaltig gespeichert und ausgewertet. In bestimmten Anwendungsgebieten stoßen relationale Datenbanken dabei an Grenzen. Gerade mit dem Anwachsen des Datenaufkommens im Internet sind neue Wege gefragt, die auch umfassenderen und loseren Datenstrukturen gerecht werden. Relationale Datenbanken leben von einem festen Schema und schränken Daten entsprechend ein. In diese Bresche springen die NoSQL-Datenbanken. Sie zeichnen sich durch Schemafreiheit und gute horizontale Skalierbarkeit aus. Anfangs noch als striktes „no SQL“ interpretiert wurde es 2009 von Johan Oskarsson im Sinne von „not only SQL“ als gemeinsamer Begriff für die wachsende Zahl an nicht relationalen, verteilten Datenspeichersystemen neu eingeführt. Gerade bei Webanwendungen wie Videoportalen und sozialen Netzwerken erfreuen sich NoSQL-Datenbanken großer Beliebtheit. Da die Daten meist aus unterschiedlichen Formaten zusammengesetzt sind, selten in vorgegebene Schemata passen und durch stetiges Wachstum hohe Skalierbarkeit erfordern.

Für die Wirtschaft sind soziale Netzwerke ein nicht versiegender Quell an Informationen und Daten. Anhand der Nutzerangaben können Zielgruppen nach Alter, Geschlecht, Heimatregion, Bildungsstand, Einkommen, Vorlieben und vielem mehr genau gefiltert und adressiert werden um den höchstmöglichen Erfolg zu erzielen. Deshalb machen Unternehmen wie Facebook riesige Gewinne indem einfach die Daten, die die Nutzer angeben, gespeichert und zur Verfügung gestellt werden. Seit dem Start von Facebook im Jahr 2004 ist die Anzahl der Nutzer stetig gestiegen.(*Abbildung 1*) Aktuell ist es mit 800 Millionen registrierten Nutzern das größte soziale Netzwerk der Welt. Täglich greifen 500 Millionen

Nutzer von ihrem Rechner und noch einmal 350 Millionen Nutzer per Handy auf Facebook zu[1], um Nachrichten zu lesen und zu senden, zu spielen, Bilder von der Party der letzten Nacht zu teilen oder zu sehen was es Neues bei den Freunden gibt. Dabei werden die Datenbanken mit einer großen Menge an Daten konfrontiert. Monatlich werden ca. 15 Milliarden Nachrichten (ca. 14 TB) und 120 Milliarden Chatnachrichten (ca. 11TB)[2] verschickt und allein am Halloweenwochenende 2011 wurden über eine Milliarde Bilder von Nutzern hochgeladen. Neben der Masse an Informationen werden bei Facebook auch verschiedene Arten von Daten gespeichert, so sind neben Texten, Bildern und Videos auch über 7 Millionen Apps in das soziale Netzwerk eingebunden. Trotz der Menge der Daten soll der Nutzer schnellstmöglichen Zugriff auf die gewünschten Informationen und Medien erhalten. Aufgrund der Menge und Varianz der Daten und der Nutzerfreundlichkeit liegt bei Facebook besonderer Fokus auf den Eigenschaften Zuverlässigkeit, Verfügbarkeit, Zugriffszeiten und Skalierbarkeit der Datenbanksysteme. Für optimale Ergebnisse wird dabei auf eine Kombination mehrerer Systeme gesetzt, um verschiedene Aufgabenbereiche abzudecken.

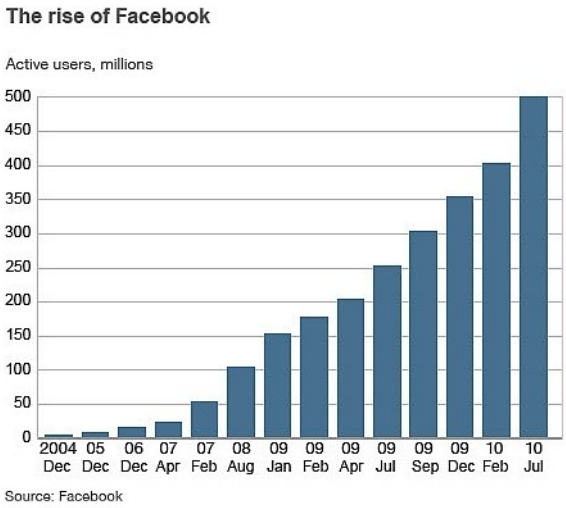

Abbildung 1: Entwicklung der Nutzerzahlen bei Facebook

In dieser Arbeit werden 3 dieser Systeme beziehungsweise deren Einsatzgebiete bei Facebook näher betrachtet. Das folgende Kapitel 2 widmet sich dabei Cassandra, einer Eigenentwicklung von Facebook, die in den Jahren 2008 bis 2011 für das Nachrichtensystem im Einsatz war. Schwerpunkt sind dabei die Eigenschaften und Architektur von Cassandra. In Kapitel 3 wird auf die Einsatzgebiete von HBase eingegangen und welche Anpassungen dafür von den Entwicklern bei Facebook vorgenommen werden musste. HBase hat unter anderem 2011 Cassandra bei der Verwaltung des Nachrichtensystems abgelöst und ist aktuell auch noch in anderen Bereichen im Einsatz. Im Kapitel 5 wird darauf eingegangen warum bei Facebook nach wie vor ein Großteil der Daten in MySQL gespeichert wird. Kapitel 6 betrachtet neben einer kurzen Zusammenfassung die Frage nach der Definition von NoSQL aus Sicht von Facebook.

2. Cassandra

2.1. Allgemeines

Cassandra wurde 2007 von den Facebookmitarbeitern Prashant Malik und Avinash Lakshman entwickelt um das „Inboxproblem“ bei Facebook zu lösen.[3] Das Ziel war Indizes von Nutzernachrichten effizient zu speichern und eine performante Suche zu ermöglichen. Hauptaugenmerk lag bei der Entwicklung auf hoher Verfügbarkeit und Skalierbarkeit. Dafür wurden Abstriche in Bezug auf die Konsistenz gemacht. Im Juli 2008 wurde Cassandra als Open Source Projekt veröffentlicht. Im März 2009 wurde Cassandra in den Apache Incubator aufgenommen und im Februar 2010 als Top-Level-Projekt geadelt.

Cassandra ist ein verteiltes Datenbanksystem, das sich durch hohe Skalierbarkeit und Verfügbarkeit auszeichnet. Durch die Verteilung auf mehrere Rechner, sogenannte Nodes, wird ein Single Point of Failure vermieden und eine bessere Lastverteilung erreicht, was bei Millionen von Nutzern und gleichzeitigen Zugriffen notwendig ist, um akzeptable Antwortzeiten zu erreichen. Aufgrund des flexiblen Datenschemas ist die Datenbank auch bei Weiterentwicklungen ständig erreichbar, was bei einer relationalen Datenbank schwer möglich ist.

Bei der Entwicklung diente Googles BigTable als Vorbild für das Datenmodell. Es wurden allerdings einige Veränderungen vorgenommen. Cassandra grenzt sich durch einen hybriden Ansatz ab, indem es sowohl Key/Value-Eigenschaften als auch eine relativ flexible Schemaunterstützung enthält. Ziel war es dabei eine größtmögliche Flexibilität und Skalierbarkeit zu ermöglichen, aber auch eine durch SQL-Datenbanken vertraute Schemasicherheit zu bieten. Vor der Version 0.7 mussten Teile der Datenbankstruktur in einer Konfigurationsdatei festgelegt werden und Änderungen zogen einen Neustart des Clusters mit sich. Ab der Version 0.7 können alle Einstellungen und Änderungen im laufenden Betrieb vorgenommen werden.[4]

Da mit Avanish Lakshman einer der beiden Entwickler zu den Autoren von Amazon Dynamo gehört, liegt es nahe, dass bewährte Eigenschaften für das aktuelle Projekt angepasst und übernommen wurden. So erinnern die Lösungen für Partitionierung, Repli-

kation, Peer-to-Peer-Erkennung und die Möglichkeit des Hinted Handoff an Amazon Dynamo.

2.2. Datenmodell

Bei Cassandra handelt es sich um einen verteilten Wide Column Store. Cassandra selbst ermöglicht keine komplexen Anfragen und Join-Kriterien, sondern nur einfache Anfragen anhand des Schlüssel der Daten. Alle Datenbankoperation darüber hinaus müssen von der Anwendung simuliert werden. Daher ist es wichtig bei der Eingabe der Daten auf leicht verständliche und intuitive Schlüssel zu achten um spätere Anfragen und Erweiterungen des „Schemas“ zu vereinfachen. Das Datenmodell orientiert sich an Googles BigTable und weist begriffliche Ähnlichkeiten mit relationalen Datenbanken auf. Dies ist in Bezug auf die Einarbeitung in das Datenmodell von Cassandra erschwerend, da der Nutzer, wenn er vorher mit relationalen Datenbanken gearbeitet hat andere Konstrukte mit diesen Begriffen verbindet. Die Hauptbestandteile des Datenmodells von Cassandra sind die folgenden Elemente. [5]

- Der **Keyspace** ist die oberste Ebene der Datenstruktur und vergleichbar mit der Datenbank bei relationalen Datenbankmanagementsystemen. Es besteht die Möglichkeit mehrere Keyspaces anzulegen, dies ist aber eher unüblich.
- In einem Keyspace werden sogenannte **Column Families** gespeichert, dabei wird eine Column Family immer in genau einem Keyspace gespeichert. Eine Column Family ist ein abstraktes Konstrukt in dem Rows gespeichert werden. Diese Rows besitzen einen Schlüssel und Columns und Super Columns als Elemente. Column Families besitzen kein vorgegebenes Schema, so dass das Wissen zur Interpretation der Daten auf Applikationsebene verbleibt. Dies stellt einen starken Kontrast zu relationalen Datenbanken dar, wo es vordefinierte Namen und Typen für die Spalten jeder Tabelle gibt. Die Namen und Werte der Columns werden als Bytes unendlicher Größe gespeichert und werden normalerweise als UTF-8 Zeichenketten oder 64bit Integerwerte interpretiert. Columns innerhalb einer Column Family werden entweder nach UTF-8-codierter Wert, als long Integer, timestamp oder einem Algorithmus der durch die Anwendung vorgegeben ist sortiert. Diese Sortierung innerhalb der Column Family ist obligatorisch und sollte je

nach Anwendungsfall gewählt werden.

- Eine **Row** ist eine eindeutig identifizierbare Einheit von Daten, die sich aus Columns und Super Columns zusammen setzt. Jede Row in Cassandra ist anhand ihres Schlüssels eindeutig identifizierbar.
- **Super Columns** sind Columns, deren Werte wiederum Columns sind. Dieses Konstrukt ermöglicht die Darstellung komplexerer Datentypen.
- **Column** ist die kleinste Informationseinheit, welche von Cassandra unterstützt wird und besteht meist aus dem Tupel (Name, Wert, Zeitstempel).

Dieses Datenmodell ermöglicht innerhalb eines groben Rahmenschemas sehr große Freiheiten wie Informationen abgelegt werden sollen. Es ist möglich während des Betriebs sämtliche Bestandteile des Systems hinzuzufügen, zu entfernen und anzupassen. Vor der Version 0.7 gab es dabei die Ausnahme, dass Keyspaces und Column Families in einer Konfigurationsdatei festgelegt werden mussten. Daher hatten Änderungen dieser beiden Elemente immer einen Clusterneustart zur Folge.

Zum näheren Verständnis des Datenmodells soll folgendes Beispiel dienen: (*Abbildung 2*) Betrachtet wird eine Anwendung zur Warenerfassung in einem beliebigen Spielzeugladen, dies ist auch gleichzeitig der Name für den Keyspace. In diesem werden alle Daten gespeichert. Analog dazu stehen die Waren auch alle im Spielzeugladen. Im Keyspace befinden sich die beiden Column Families „Spielwaren“ und „sonstiges“. Um bei der Analogie zu bleiben könnten eine Abteilung oder ein Regal im realen Spielzeugladen darstellen. Diese Column Families fassen verschiedene Rows zusammen, die durch einen Key eindeutig identifizierbar sind. Bei den Spielsachen bildet die Column Family beispielsweise ein Regal ab und die Rows sind die klar voneinander getrennten Regalböden in denen sich jeweils genau 1 Spielzeug befindet. So befinden sich in der einen Row die Informationen zu einem Auto und in der anderen zu einer Puppe. Diese Informationen werden in Columns gespeichert, in denen Name und Wert der Information gespeichert sind. Das Auto hat die Eigenschaften „Farbe“ und „Preis“, während bei der Puppe die Eigenschaften „Typ“, „Name“ und „Preis“ hinterlegt sind. Die Eigenschaften können also für jeden Artikel im System vollkommen frei gewählt werden.

In der Column Family „sonstiges“ sind alle sonstigen Waren des Geschäfts verzeichnet. Um die Einträge übersichtlicher zu gestalten wurden dafür Supercolumns verwendet. Im

Beispiel handelt es sich dabei um „Süßigkeiten“ und ist vergleichbar mit einem Süßigkeitenregal. Hier befindet sich dann nicht pro Regalboden genau ein Artikel, sondern mehrere. Allerdings sorgt die Aufteilung der Regalböden in „Schokolade“, „Bonbons“ und „Kaugummi“ für mehr Übersichtlichkeit.

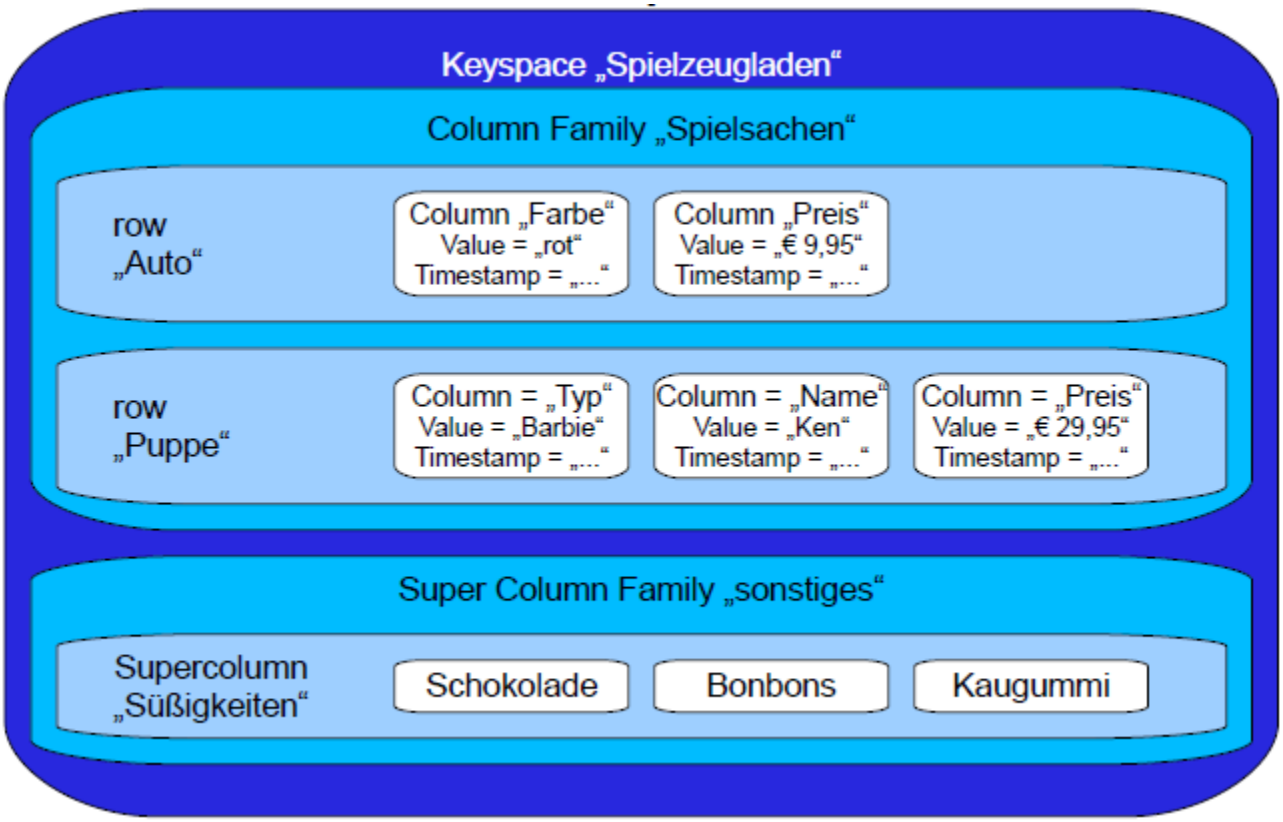

Abbildung 2: Beispiel „Spielzeugladen“ für das Datenmodell

2.3. Konsistenz

Bei Cassandra nutzt, wie bei die meisten NoSQL-Datenbanksysteme, das Konsistenzmodell der „eventual consistency“. Die Daten sind zu einem bestimmten Zeitpunkt konsistent, bis dahin können die anderen Clients entweder noch auf die alten Daten zugreifen oder der Zugriff ist aufgrund einer Sperre nicht möglich. Während bei vielen Datenbanksystemen festgelegt ist nach welchen Kriterien die Daten als konsistent gelten gibt es bei Cassandra die Möglichkeit zwischen verschiedenen Konsistenzleveln zu wählen. Diese orientieren sich am CAP-Theorem, nach dem in verteilten Systemen von den 3 Punkten Consistency, Availability und Partition Tolerance nur jeweils 2 Kriterien vollständig erfüllt werden können.

Bei Cassandra stehen die folgenden Konsistenzlevel zur Verfügung:

- **Zero:** Es gibt keine Garantie, dass die Schreiboperation erfolgreich ausgeführt wurde. Es gilt die optimistische Annahme, dass die Daten aktuell sind, wenn darauf zugegriffen wird. Diese Einstellung erreicht die höchste Verfügbarkeit (Availability) auf Kosten der Konsistenz, die nicht garantiert werden kann. Im worst case gehen die Daten verloren und sind nicht rekonstruierbar.
- **Any:** Mindestens ein Speichernode muss das erfolgreiche Schreiben der Daten bestätigen, damit die Konsistenz als gegeben gilt. Die Daten müssen dafür nicht zwingend auf dem eigentlichen Zielnode gespeichert werden. Bei dieser Variante ist Hinted Hintoff möglich. Dabei springt bei einem Ausfall des Zielnodes ein Nachbarnode ein, der sich die Änderungen merkt und diese an den eigentlichen Adressaten weiterleitet sobald dieser wieder verfügbar ist. Dadurch besteht weiterhin die Möglichkeit, dass bei einer Anfrage veraltete Daten gelesen werden, da die Änderungen noch nicht auf allen Replikanodes durchgeführt sein müssen.
- **One:** Bei Schreiboperationen wird sichergestellt, dass mindestens ein Node die Änderungen im Commit Log und in die RAM-Tabellen geschrieben hat, bevor der Client eine Bestätigung bekommt. Bei einem Lesezugriff wird wird dem Client das erste verfügbare Ergebnis zurückgegeben, welches auch bei dieser Einstellung noch aus veralteten Daten bestehen kann.
- **Quorum:** Bei Schreiboperationen müssen *Replikationsfaktor R / 2 +1* Replikate die Operation bestätigen, damit der Client eine positive Rückmeldung bekommt. Beim Lesen wird der Eintrag mit dem letzten Timestamp zurückgegeben. Mit dieser Strategie werden Inkonsistenzen während der Synchronisationsphase nach außen hin verborgen und die Daten sind immer aktuell.
- **All:** Die Schreiboperation gilt für den Client erst als abgeschlossen, wenn alle Replikanodes die Änderungen bestätigt haben. Diese Variante ist sehr teuer, da hohe Konsistenz mit niedriger Verfügbarkeit erkauft wird. Daher sollte sie nur bei Daten verwendet werden, die auf gar keinen Fall Inkonsistenz aufweisen dürfen, wie zum Beispiel Kontodaten.

Abbildung 3 zeigt noch einmal eine Übersicht über alle Konsistenzlevel. Mit diesen 5 verschiedenen Konsistenzleveln bietet Cassandra dem Nutzer die Möglichkeit Konsistenz und Verfügbarkeit an die Bedürfnisse der Anwendung anzupassen. Dabei gilt zu beachten, dass hohe Verfügbarkeit immer auf Kosten der Konsistenz erreicht wird und die Verfügbarkeit durch hohe Konsistenz zum Teil erheblich eingeschränkt werden kann.

Schreiben		Lesen	
Level	**Beschreibung**	**Level**	**Beschreibung**
ZERO	„Hail Mary!“	ZERO	N/A
ANY	1 Knoten (HH)	ANY	N/A
ONE	1 Knoten	ONE	1 Knoten
QUORUM	(N/2)+1	QUORUM	(N/2)+1
ALL	Alle Knoten	ALL	Alle Knoten

Abbildung 3: Übersicht über die Konsistenzlevel bei Cassandra

2.4. Architektur

Bei der Entwicklung von Cassandra spielte die Skalierbarkeit und Robustheit des Systems eine große Rolle. Im Idealfall sollen die Daten in einem Cluster jederzeit verfügbar sein und auch unter großer Last abgefragt werden können. Um dies zu erreichen nutzt Cassandra das Prinzip der Distributed Hash Table. Dabei werden die Daten über Schlüssel adressiert, die sich über einen Adressraum fester Größe verteilen. Bei Cassandra hat dieser die Größe von 127 Bit. Der Schlüsselraum wird gleichmäßig über alle Nodes des Clusters verteilt, so dass unabhängig vom physikalischen Netzwerk eine logische Ringstruktur entsteht. Dank Consistent Hashing kann jeder Server des Clusters jederzeit anhand eines gegebenen Schlüssels bestimmen, welcher Node aktuell für diesen Schlüssel zuständig ist und die Anfragen direkt weiterleiten.

Um die Robustheit des Systems zu sicherzustellen werden von allen Daten, die in Cassandra gespeichert werden, Replikate angelegt. Die Anzahl richtet sich dabei nach dem Replikationsfaktor N, den der Nutzer bestimmt. Die Hashingfunktion listet nur die Speicheradresse des Nodes, der primär für die Daten zuständig ist. Jeder Knoten besitzt

selbst eine Liste mit N-1 anderen Knoten, auf denen die Kopien gespeichert sind. Diese Informationen werden unter den einzelnen Nodes ausgetauscht, so dass die Adressen der Replikate bekannt sind. Wenn ein Knoten Daten schreiben oder lesen soll greift er auf das nächste Replikat zu, sollte der primäre Node nicht erreichbar sein. Ist ein Knoten wieder zurück im Netzwerk werden die aufgelaufenen Writes synchronisiert um einen konsistenten Zustand herzustellen. Replikate können bei Cassandra auf verschiedene Weise angelegt werden. Es werden 2 Strategien angeboten, abhängig davon ob die logische oder die physikalische Netzwerktopologie genutzt werden soll um die Replikate zu verteilen. Bei der rack unaware-Strategie handelt es sich um die Standardmethode für Replikation in Cassandra. Hier wird die physikalische Verteilung vernachlässigt und die Daten werden beginnend bei dem primären Knoten auf N-1 folgende Nodes kopiert. Die *Abbildung 4* zeigt ein Beispiel dieser Strategie mit einem Replikationsfaktor von N=3. Die zweite Möglichkeit ist die rack aware-Strategie. Bei dieser Variante wird eine möglichst gute Ausfallsicherheit dadurch erreicht, dass die Daten auf verschiedene Serverschränke und Rechenzentren verteilt werden. Die Daten wären auch nach dem Ausfall eines kompletten Rechenzentrums weiterhin verfügbar. Dafür werden die die IP-Adressen der einzelnen Nodes verwendet. Cassandra geht dabei davon aus, dass es sich bei Abweichungen beim 2. Oktett der IP-Adresse ebenfalls um verschiedene Rechenzentren handelt, analog bei Abweichungen beim 3. Oktett und Serverschränken. Die Strategie sieht vor, dass das erste Replikat in einem anderen Rechenzentrum und das zweite in einem anderen Serverschrank liegt. Die verbleibenden N-3 Kopien werden mit der rack unaware-Strategie verteilt. Darüber hinaus hat der Nutzer die Möglichkeit eigene Strategien für die Replikation festzulegen.[5]

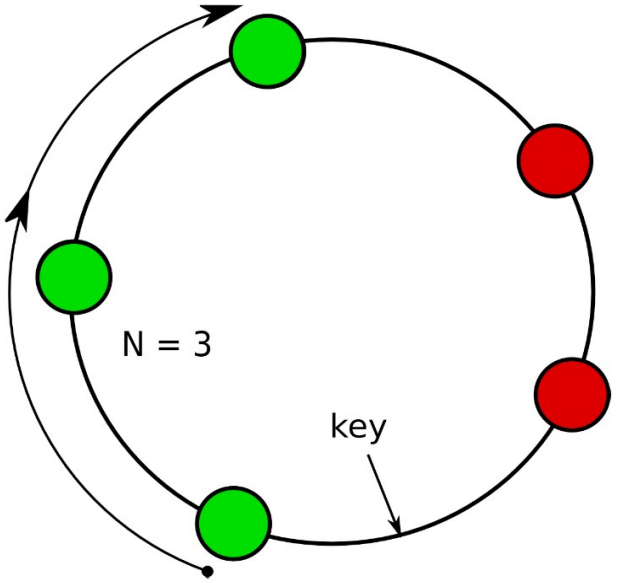

Abbildung 4: logische Ringstruktur eines Clusters (Replikationsfaktor N=3)

Ein Hauptargument für Cassandra ist die einfache Skalierbarkeit. Es ist mit sehr geringem Aufwand möglich Knoten hinzuzufügen und zu entfernen. Cassandracluster skalieren mit

der Anzahl der Rechner und es entstehen keine hohen Kosten durch den Erwerb von leistungsfähigeren Servern. Diese Art der Skalierung wird horizontale Skalierung genannt, im Gegensatz zu vertikaler Skalierung, bei der in immer teurere Technik investiert werden muss. Das Hinzufügen neuer Nodes in Cassandra geschieht via Bootstrapping auf eine von 2 Möglichkeiten. Bei der ersten Möglichkeit meldet sich der neue Knoten bei einem bestimmten anderen Knoten im Netzwerk an. Dieser übergibt ihm einen Token, der seine Adresse im logischen Ring festlegt. Danach migriert ein Teil der Daten des nächsthöheren Knotens auf den neuen Knoten, so dass eine Lastverteilung stattfindet. Die *Abbildung 5* zeigt ein Beispiel für Bootstrapping. Dem neuen Knoten wurde eine Adresse zwischen den Nodes n und n+1 zugeteilt und ein Teil der Daten von n+1 wird auf den neuen Knoten verschoben.[5]

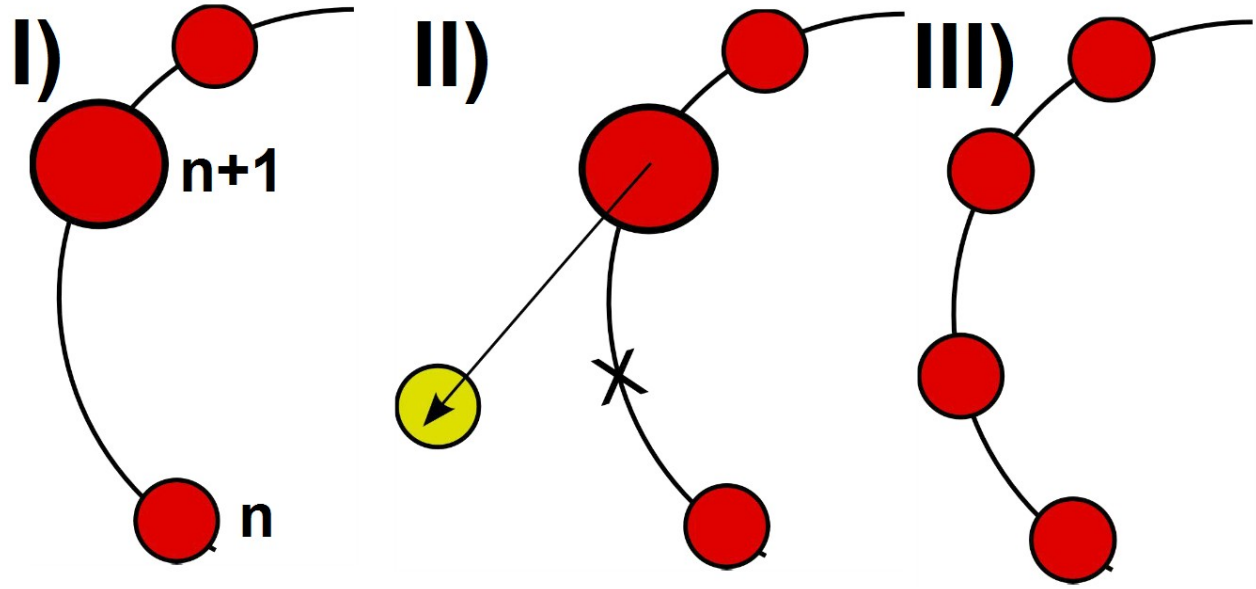

Abbildung 5: Bootstrapping bei Hinzufügen eines neuen Knotens

Bei der zweiten Möglichkeit wählt der Cluster den Token aus, der die Adresse des neuen Nodes bestimmt. Das Ziel dabei ist es den Knoten, der aktuell die höchste Auslastung hat, zu entlasten. Auch wenn es sich bei diesem Verfahren praktisch um eine Abstimmung handelt, werden nicht alle Knoten des Clusters kontaktiert. Der neue Node entscheidet aufgrund der Auslastung des Speichers der einzelnen Knoten, welche im Netzwerk kommuniziert wird. Mit diesem Verfahren kann durch das Hinzufügen neuer Rechner in einem Cluster das Load-Balancing verbessert werden.[5]

Das dauerhafte Entfernen eines Knotens ist nicht mit dem Entfernen des Rechners vom Netzwerk getan, da die Replikaknoten nur dessen Ausfall verzeichnen. Sie speichern

weiterhin die Änderungen für den Knoten um ihm diese später mitzuteilen. Cassandra bietet dafür ein Clustermanagementtool an, mit dem Knoten dauerhaft entfernt werden können und das deren Daten neu verteilt. Diese Art der Rückskalierung fehlt bei einigen anderen NoSQL-Datenbanken.

2.5. Vor- und Nachteile

Cassandra wurde seit seiner Veröffentlichung als Open Source Projekt konsequent weiterentwickelt. Dabei kam es zwischen einzelnen Versionen zu teilweise sehr großen Änderungen. So rückte mit der Version 0.7 der Stellenwert der Konfigurationsdatei in den Hintergrund. Mit der Version 0.8 bekam der Nutzer mit CQL eine vertraute Abfragesprache aus dem SQL-Umfeld spendiert. Bis zur Version 1.0 wurde unter anderem die Komprimierung der Daten und die damit zusammenhängenden Zugriffszeiten optimiert, die Performance von Lesezugriffen während größerer Änderungen am Datenbestand verbessert und die Version 1.1 von CQL implementiert. CQL 1.1 bietet als zusätzliche Funktionen unter anderem eine ALTER-Funktion, den Support von Countern, Time to Live-Support und prepared Statements. Für die zukünftigen Versionen ist hauptsächlich die Verbesserung der Benutzerfreundlichkeit geplant.

Die **Vorteile** von Cassandra sind

- die extrem einfache Skalierbarkeit. Für das Hinzufügen von Nodes muss nur eine neue Instanz von Cassandra gestartet werden. Weitere Aktionen sind nicht notwendig. Genauso einfach lässt sich der Ring auch wieder verkleinern. Es gibt keinen Single Point of Failure und das Schreiben und Lesen von Daten ist nicht auf bestimmte Nodes beschränkt. Cassandra-Cluster sind selbst heilend und können auf wenig Latenz optimiert werden.
- eine eigene Abfragesprache CQL, die dem Nutzer aus dem SQL-Umfeld vertraut ist.
- Erweiterung des Datenmodells in einem vorgegebenen Rahmen sind im laufenden Betrieb möglich und erfordern ab der Version 0.7 keinen Clusterneustart mehr. Innerhalb der Column Families herrscht die selbe Freiheit wie bei einer Dokumentdatenbank. Neue Key/Value-Paare können zur Laufzeit beliebig eingefügt werden.

Mit den Supercolumns existiert die Möglichkeit Columns als Listen von Key/Value-Paaren zu nutzen.

- effiziente Bereichsanfragen über indizierte Daten. Dabei können nicht nur über Schlüssel eines Datensatzes Indizes gebildet werden, sondern, in gewissen Grenzen, auch über beliebige Columns.
- detailliert konfigurierbare Replikationsstrategien, die eine Verteilung der Daten über mehrere Standorte erlauben.
- gute Konfigurierbarkeit bezüglich des CAP-Theorems. Je nach Anforderungen kann der Anwender bestimmen in welcher Ecke des CAP-Dreiecks das System eher liegen soll.

Als **Nachteile** gelten

- die geringe Menge an Abfragemöglichkeiten im Vergleich mit anderen Datenbankmanagementsysteme wie zum Beispiel Hadoop. Für Map/Reduce-ähnliche Berechnungen können über die Cassandra-Hadoop-Inspiration nachgerüstet werden.
- die Notwendigkeit von synchronisierten Uhren zur Konfliktbehandlung. Es besteht daher eine Abhängigkeit zu einer funktionierenden Zeitsynchronisation im Cluster.
- eine sehr verstreute Dokumentation. Wer auf frei verfügbare elektronische Quellen zurückgreifen will, muss einiges an Zeit für die Recherche einplanen. Auch gibt es gerade zum Datenmodell immer wieder widersprüchliche Angaben, die es schwer verständlich und nachvollziehbar machen. Aufgrund der schnellen Weiterentwicklung, dem Release neuer Versionen und den damit verbundenen zum Teil sehr starken Änderungen ist ein Großteil der Literatur sehr schnell veraltet und enthält dadurch falsche Informationen. Das erschwert die Einarbeitung in das Thema erheblich, da meist nicht sofort nachvollziehbar ist auf welche Version von Cassandra sich die Ausarbeitung bezieht.[4]

Cassandra ist vor allem bei Onlineplattformen im Einsatz und erfreut sich großer Beliebtheit. Prominente Nutzer sind zum Beispiel Netflix, die von einem Datacenter mit Oracle zu einem globalen Netzwerk mit Cassandra gewechselt haben, Twitter, wo Cassandra unter anderem zur Analyse des Inhalts der geposteten Tweets nutzen und

Digg, deren Datenbank komplett auf Cassandra basiert. Bei Expedia wird Cassandra genutzt um Preise und Daten von Hotels in Echtzeit zu empfangen und zu speichern, bei Adobe wird es im Adobe Audience Manager genutzt und Cisco's Webex speichert Nutzereingaben und -aktivität in Cassandra. Bei Facebook allerdings wurde es 2010 mit der Einführung des neuen Nachrichtensystems von HBase abgelöst.

3. HBase

3.1. Allgemeines

HBase ist eine skalierbare, einfache Datenbank zur Verwaltung sehr großer Datenmengen innerhalb eines Hadoop-Clusters. Die HBase-Datenbank basiert auf einer freien Implementierung von Google BigTable. Diese Datenstruktur ist für Daten geeignet, die selten verändert werden, dafür aber sehr häufig ergänzt werden, womit HBase sich für das Nachrichtensystem bei Facebook geradewegs anbietet. Mit HBase lassen sich Milliarden von Zeilen verteilt und effizient verwalten.[6]

Das Hadoop Distributed File System (HDFS) ist ein hochverfügbares, leistungsfähiges Dateisystem zur Speicherung sehr großer Datenmengen. Im HDFS gibt es zwei Typen von Servern (Clusternodes): NameNodes und DataNodes. Die Datenblöcke werden auf den DataNodes abgelegt, sämtliche Meta-Informationen sind auf den NameNodes gespeichert. Dateien werden dabei auf mehrere Datenblöcke verteilt. Zur Steigerung der Zuverlässigkeit und Geschwindigkeit legt HDFS mehrfach Kopien von einzelnen Datenblöcken an. Ein Masterknoten (NameNode) bearbeitet eingehende Datenanfragen, organisiert die Ablage von Dateien und speichert anfallende Metadaten. HDFS unterstützt Dateisysteme mit mehreren 100 Mio. Dateien. [7]

3.2. Nachrichtensystem als Haupteinsatzgebiet

Im November 2010 wurde bei Facebook ein neues Nachrichtensystem eingeführt. Neben den bisherigen internen Nachrichten, die sich die Nutzer gegenseitig schicken konnten, umfasst das neue System zusätzlich den Chat, extern verfasste Emails und SMS, die Nutzer über Facebook empfangen können. Im Zuge dieser Änderungen sollte gleichzeitig die Dateninfrastruktur an das neue Nachrichtensystem angepasst werden. Um zu prüfen welche Änderungen den größten Nutzen mit sich bringen, wurde intern die Art und das Verhalten der Nachrichten zwischen den Anwendern untersucht. Die Analyse der Daten ergab, dass hauptsächlich 2 Arten von Nachrichten auftreten[8]:

- „A short set of temporal data that tends to be volatile“
- „An ever-growing set of data that rarely gets accessed“

Nur ein kleiner, sich ständig ändernder, Teil der Informationen wird regelmäßig abgerufen, während die Menge der Daten ständig größer wurde, aber kaum Zugriffe darauf stattfanden. Das Volumen der Daten betrug beim alten Nachrichtensystem 15 Milliarden Nachrichten und 120 Milliarden Chatnachrichten monatlich. Das entspricht ca. 14 bzw. 11 Terabyte an Daten.[2] Anhand dieser Informationen und Daten wurden bei Facebook Tests mit Cassandra, HBase und MySQL durchgeführt, welches Datenbankmanagementsystem für diese Art von Zugriffen am besten geeignet sei. Als Ergebnis dieser Tests wurde HBase als Sieger und erste Wahl für das neue Nachrichtensystem vorgestellt. Als Begründung zur Entscheidung gegen die Konkurrenten wurde genannt, dass MySQL „Probleme mit dem langen Datenschwanz“ bekommt und bei Cassandra das Konsistenzmodell zu komplex ist. Bei der Begründung der Wahl für HBase wurden mehrere Gründe genannt. HBase verfügt über eine sehr gute Skalierbarkeit, einen sehr hohen Schreibdurchsatz, ein effizientes und schnelles Konsistenzmodell und eine gute Fehlertoleranz und -isolation auf Speicherebene.[8][9] Ein Vorteil direkt gegenüber Cassandra ist die Unterstützung von Ranged Scans, die in Cassandra nur auf Indizes möglich ist, um Nachrichten zum Beispiel nach Zeiträumen, Absendern oder den letzten xx Nachrichten zu filtern. Ein weiterer Vorteil ist der schnelle Zugriff bei „random reads“, falls Nutzer doch auf ältere Nachrichten zurückgreifen wollen. Darüber hinaus bringt HBase „build in features“ wie eine automatische Lastverteilung, Datenkompression und „read-write-modify support“ mit, so dass diese nicht erst implementiert werden mussten. Gerade der letzte Punkt mit der Möglichkeit zu sehen ob Nachrichten gelesen oder geändert wurden macht speziell bei einem Nachrichtensystem Sinn.

Ein großer Nachteil bei HBase war der NameNode. Auf diesem sind sämtliche Metadaten gespeichert und bei einem Ausfall wäre der Cluster nicht mehr erreichbar. Damit ist eine durchgängige Erreichbarkeit nicht gewährleistet. Für dieses Problem wurde intern bei Facebook mit dem AvatarNode eine Lösung entwickelt, auf die im Kapitel *3.3. Anpassungen* näher eingegangen wird.

In HBase werden die Metadaten, Textinhalte und Suchindizes gespeichert und verwaltet. Bei Anhängen, Bildern und sehr langen Textnachrichten kommt mit Haystack eine weitere Eigenentwicklung von Facebook zum Einsatz. Aktuell (Stand Ende 2011) verwaltet HBase 6 Milliarden Nachrichten am Tag, bewältigt mehr als 75 Milliarden Lese- und Schreiboperationen pro Tag und bis zu 1,5 Millionen Operationen pro Sekunde. Davon sind ca.

55% Lese- und 45% Schreibzugriffe, die einen Umfang von bis zu 16 Einträgen in unterschiedlichen Column Families haben. Das Datenvolumen der Nachrichten in HBase bei über 2 Petabyte, mit Replikaten sogar bei über 6PB, da jede Nachricht zum Schutz vor Verlust 3fach gespeichert wird. Der monatliche Zuwachs beträgt in etwa 250TB.[2]

3.3. Anpassungen

Der NameNode stellte einen Single Point of Failure dar. NameNodes sind grundsätzlich redundant ausgelegt, jedoch als Aktiv/Passiv-Cluster organisiert. Fällt der aktive NameNode wegen eines Hardwarefehlers aus, muss der passive gestartet werden und die „Block Records“ aller Nodes einsammeln um die Speicherort aller Datenblöcke zu erlernen. Der größte HDFS-Cluster bei Facebook umfasst ca. 150 Millionen Dateien, was eine Startzeit von 45 Minuten zur Folge hätte. Während dieser Zeit ist kein Zugriff auf das HDFS möglich. Aktuell laufende Schreibprozesse auf das Filesystem werden mit einer Fehlermeldung abgebrochen. Die Nutzer sollen aber jederzeit ihre Nachrichten abrufen können und von eventuellen Serverausfällen nichts mitbekommen. Daher haben die Entwickler bei Facebook den AvatarNode eingeführt. Dabei handelt es sich um einen Active AvatarNode und einen Standbye AvatarNode, die beide Informationen von den DataNodes erhalten und damit aktuell bleiben. Fällt der Active AvatarNode aus kann der Standbye AvatarNode innerhalb von Sekunden einspringen.[10]

Darüber hinaus wurden noch zahlreiche zusätzliche Anpassungen vorgenommen um die Zuverlässigkeit, Verfügbarkeit und Leistung von HBase zu verbessern.[9] Zur Verbesserung der Zuverlässigkeit gab es mehrere Bugfixes bei der Logrecovery, es wurde die Unterstützung von HDFS sync und eine neue Block-Placement-Policy implementiert, sowie atomare Multi-Column Family-Transaktionen. Bei HDFS sync werden Daten aus dem Schreibbuffer des Clients in einer Datenpipeline gespeichert und können dort von anderen Clients gelesen werden. So gehen die Daten nicht verloren, wenn während des Schreibvorgangs Client oder DataNode ausfallen. Bei der neuen Block-Placement-Policy ging es den Entwicklern bei Facebook darum, dass sie einen Einfluss darauf haben, wo bestimmte Daten gespeichert werden sollen. So sollen zum Beispiel die Nachrichten eines Nutzers alle im selben HDFS-Cluster liegen. Bei der Verfügbarkeit ging es hauptsächlich darum, die Ausfallzeiten bei Arbeiten am System so gering wie möglich zu halten bzw.

ganz zu eliminieren. So werden Softwareupdates via „rolling Upgrades“ durchgeführt, bei denen nicht alle Server gleichzeitig gepatcht werden, sondern nacheinander. Der Ausfall einzelner Server lässt sich durch die Verteilung der Daten kompensieren. Eine weitere Verbesserung ist, dass Schemaänderungen online durchgeführt werden können. Bei Clusterneustarts und Lastenumverteilung kam es zu enormen Wartezeiten, was daran lag dass die Datenkompression erst beendet werden musste, bevor diese ausgeführt wurden. So musste eine Möglichkeit implementiert werden diese Datenkompression zu unterbrechen um diese Wartezeiten zu umgehen. In Bezug auf die Leistung wurde an einer Suchoptimierung, Multigets, einer besseren Verarbeitung komprimierter HFiles und an Bloomfiltern gearbeitet. Bloomfilter sind Datenstrukturen um schnell festzustellen welche Daten in einem Datenstrom schon vorgekommen sind und welche erstmals Auftreten. Damit lassen sich Anfragen an die Datenbank vermeiden indem „heiße Daten“ im Cache gespeichert werden. Die Mitarbeiter von Facebook haben demnach einiges an Arbeit investiert um das bestehende HBase an die Anforderungen ihrer Plattform bestmöglich anzupassen.

3.4. Weitere Einsatzgebiete

Neben dem Nachrichtensystem gibt es bei Facebook noch zwei weitere Anwendungsfälle für HBase. Facebook Insights bietet Entwicklern und Besitzern einer Website Zugriff auf die Analyse von Aktivitäten der Nutzer auf Seiten mit Facebook-Plugin, Facebookseiten und Werbung bei Facebook. Diese Daten werden anonymisiert und sind in Echtzeit abrufbar. Es werden unter anderem „Likes“, Browseverhalten des Nutzers und die Anzahl der Aufrufe einer Seite gespeichert. Diese Daten sollen es Firmen, aber auch Bloggern, ermöglichen das Verhalten der Besucher ihrer Seite zu analysieren und dabei helfen ihren Internetauftritt entsprechend anzupassen.[10]

Das Facebook Metrics System speichert alle hard- und softwarespezifischen Daten, die im laufenden Betrieb anfallen im Operations Data Store (ODS). Dabei kann es sich um die CPU-Auslastung eines Servers handeln oder um die Anzahl der Lese- und Schreibzugriffe auf einen HBase-Cluster. Für jeden Knoten werden hunderte bis tausende Daten erfasst, diese können mit unterschiedlicher Granularität dargestellt werden. Diese Informationen sind wichtig für die Entwickler bei Facebook, um z.Bsp. gezielt Nodes zu entlasten.[10]

4. MySQL

4.1. Allgemeines

MySQL ist mit mehr als 6 Millionen Installationen und über 65.000 Downloads pro Tag das am weitesten verbreitete relationale Datenbankmanagementsystem. Es ist als Open Source-Software sowie als kommerzielle Enterpriseversion für verschiedene Betriebssysteme verfügbar und bildet die Grundlage für viele dynamische Webauftritte. MySQL wurde seit 1994 vom schwedischen Unternehmen MySQL AB entwickelt. Im Februar 2008 wurde MySQL AB von Sun Microsystems übernommen, das selbst im Januar 2010 von Oracle gekauft wurde.

Ein bevorzugtes Einsatzgebiet von MySQL ist die Datenspeicherung für Webservices. MySQL wird dabei häufig in Verbindung mit dem Webserver Apache und der Skriptsprache PHP eingesetzt. Bei Facebook ist eine Vielzahl von MySQL-Servern im Einsatz, über die die Zugriffe aus dem Netz abgewickelt werden.[11]

4.2. Einsatz von MySQL bei Facebook

Bei Facebook werden nahezu alle Daten in MySQL gespeichert. Dabei handelt es sich zum Beispiel um Nutzerdaten, Statusupdates, „Likes“, Warnungen und Anfragen. Da es sich bei Facebook um ein soziales Netzwerk handelt hat jede Aktion Auswirkungen auf mehrere Nutzer. Offensichtlich ist das zum Beispiel bei Anfragen, Beziehungen und Nachrichten auf der Profilseite anderer Nutzer, aber auch eigene Statusupdates und Veröffentlichungen haben Einfluss auf die Daten der Freunde, da diese über die Änderungen informiert werden „müssen“, wenn sie die Startseite aufrufen. Deshalb betrifft jede Aktion eines Nutzers immer mehrere Datensätze. Dadurch müssen zu Stoßzeiten bis zu 60 Millionen Anfragen und 4 Millionen Zeilenoperationen pro Sekunde bearbeitet werden (Stand 2010). Um das hohe Datenaufkommen und die Masse an Anfragen in akzeptabler Zeit zu bewältigen kümmert sich bei Facebook ein ganzes Team nur darum MySQL zu verbessern um es performanter zu machen. Ein Schritt zu besserer Performance war das Aufteilen der Nutzerdatenbank in mehrere Sektionen, so dass mehrere

Informationen parallel abgerufen werden können. Der größte Gewinn wird durch das Bereithalten sehr vieler Daten im Cache erreicht. Mehr als 90% der Anfragen erreichen nie die Datenbank, sondern werden vom Cache-Layer abgefangen und beantwortet. Dafür wird eine Kombination von memcached und der Eigenproduktion Flashcache eingesetzt, die es ermöglicht Daten auf SSD-Laufwerken zu cachen. Begünstigend wirkt sich dabei aus, dass viele ältere Einträge der Nutzer so gut wie nie abgerufen werden und sich der Fokus meist auf die neuesten Entwicklungen beschränkt. Frei nach dem Pareto-Prinzip betreffen 80% der Anfragen nur 20% der Daten.[12]

Ein weiterer Faktor ist die ständig wachsende Menge an Daten. Stand Ende 2011 hat sich die Serverkapazität in den letzten beiden Jahren verdreifacht. Die gespeicherten Nutzerdaten sind auf das siebenfache, mit Replikaten sogar auf das zwanzigfache gestiegen und ein Nutzerdatensatz ist im Durchschnitt 5x (maximal 10x) so groß wie noch vor 2 Jahren. Die MySQL Nutzerdatenbank verteilt sich sich zu ungefähr 60% auf HDD, 20% auf SSD und 10% auf SSD-HDD-Hybridlösungen mit Flashcache.[12] Daraus ergibt sich ein großer finanzieller Aspekt, da die Ausgaben für Server sehr hoch ausfallen. So lief Mitte 2011 die MySQL-Datenbank über 4000 Shards verteilt und zusätzlich 9000 Instanzen von memcached um die Menge an Daten zu bewältigen.[13] Ziel ist es diese Ausgaben zu verringern. Maßnahmen dafür sind die Automatisierung der Aufteilung großer Datensätze auf wenig genutzte Server, eine Verbesserung der Datenkompression in SQL und die Migration eines Teils der Daten nach HBase.

Trotz der Probleme und Kosten bekennt sich Facebook weiterhin zu MySQL. Auf die Aussage des Datenbankpioniers Michael Stonebraker, dass das Schicksal der MySQL-Implementierung bei Facebook „worse than death“[13] sei, konterte Facebook mit der Veröffentlichung der aktuellen Leistungsdaten. Diese sind in der *Abbildung 6* aufgelistet. In dem Vortrag wurden auch weitere Gründe für das Festhalten an MySQL aufgelistet. So spart die Verwendung von Open Source-Software Geld für Lizenzen und Mitarbeiter, da die Unterstützung durch die Open Source-Community Ressourcen spart, die für die Arbeit an anderen Projekten verwendet werden können. Ein weiterer Vorteil sind die geringen Entwicklungszeiten bei Patches und Updates, da diese im Haus entwickelt werden können und keine Abhängigkeit von anderen Firmen und deren Produktzyklen entsteht. Bei Problemen kann sofort auf Personal vor Ort zurückgegriffen werden, ohne dass lange Wartezeiten entstehen. Ohnehin arbeitet bei Facebook das „beste MySQL-Team“.[12]

	2009	September 2011
query response time	4ms Lesen, 5ms Schreiben	4ms Lesen, 5ms Schreiben
Netzwerklast / Sekunde	Max 38GB	Max 90GB
Anfragen / Sekunde	Max 13 Millionen	Max 60 Millionen
Einträge lesen / Sekunde	Max 450 Millionen	Max 1450 Millionen
Einträge ändern / Sekunde	Max 3,5 Millionen	Max 3,5 Millionen
Plattenoperationen / Sekunde	Max 5,25 Millionen	Max 8,1 Millionen

Abbildung 6: Leistungsdaten von MySQL im Einsatz bei Facebook

Grundsätzlich ist man nach eigenen Angaben bei Facebook nicht abgeneigt neue Wege zu gehen was das Datenbankmanagementsystem angeht. Dem im Weg steht aber die Möglichkeit kommerzielle Software vorher ausgiebig zu testen und die Ergebnisse öffentlich zu machen. Darauf besteht Facebook, da ein Einsatz in der Größenordnung für eine Datenbank nicht alltäglich ist. NoSQL-Datenbanken wiederum haben noch nicht die nötige Reife um in dieser Größenordnung zuverlässig zu operieren. Darüber hinaus benötigt man für HBase ein größeres Team als für MySQL, was gleichzeitig höhere Kosten bedeutet. Deshalb wird auch in Zukunft auf MySQL gebaut bei Facebook, auch wenn das einstige Monopol zu bröckeln beginnt und andere Datenbanksysteme wie HBase oder zwischenzeitlich Cassandra Einzug halten.

5. Zusammenfassung

Als soziales Netzwerk stellt Facebook besondere Anforderungen an die Datenbankmanagementsysteme. Diese werden benötigt um die Menge an verschiedenen Daten zu erfassen, zu speichern, zu analysieren und zur Verfügung zu stellen. Das Hauptaugenmerk liegt dabei auf den Punkten Zuverlässigkeit, Verfügbarkeit und Skalierbarkeit bei für den Anwender akzeptablen Zugriffszeiten. Um diesen Anforderungen gerecht zu werden gibt es keine Konzentration auf eine spezielle Technologie, die in allen Bereichen Anwendung findet. Es werden eine Vielzahl von Datenbanktechnologien kombiniert. Dabei kommen mit MySQL bewährte Systeme zum Einsatz, die von den Mitarbeitern dauerhaft weiterentwickelt und verbessert werden. Darüber hinaus werden bestehende Lösungen, wie HBase, an die speziellen Anforderungen und Aufgabenbereiche bei Facebook angepasst und in die bestehende Infrastruktur integriert. Sollten existierende Lösungen nicht ausreichen wird auf Eigenentwicklungen, wie Haystack oder Cassandra zurückgegriffen. MySQL ist seit dem Start von Facebook im Jahr 2004 dauerhaft im Einsatz. In den Jahren 2008 bis 2011 wurde Cassandra für das Nachrichtensystem eingesetzt. 2011 wurde von Cassandra auf HBase zur Verwaltung des Nachrichtensystems umgestellt. NoSQL-Datenbanken sind nicht DIE Lösung für das Problem der großen Datenmengen, bieten aber meist sehr gute Insellösungen für spezielle Probleme. Bei Facebook steht NoSQL demnach für „not only SQL“.

Interessant für die Zukunft ist die Frage wie lange sich die bisher verwendeten Technologien und dabei speziell MySQL weiterhin tunen lassen um den Anforderungen gerecht zu werden. Mit der Umstellung der Nutzerprofile auf die neue Chronik müssen ältere Einträge und Daten schnell gefunden und dargestellt werden. Das erschwert die momentane Lösung des Cachens der Daten auf die am wahrscheinlichsten zugegriffen wird. Mit dieser Frage konfrontiert war die Antwort, dass man optimistisch sei dieses Problem in den Griff zu bekommen, man könne aber noch keine Lösung präsentieren.

6. Quellen

1: fbtechtalks - MySQL & HBase, 2011-11, http://www.livestream.com/fbtechtalks/video?clipId=pla_a3d62538-1238-4202-a3be-e257cd866bb9&utm_source=lslibrary&utm_medium=ui-thumb, Letzter Aufruf: 01.04.2012
2: The Underlying Technology of Messages Tech Talk, 2010-07, http://www.facebook.com/photo.php?v=690851516105&set=vb.9445547199&type=2&theater, Letzter Aufruf: 01.04.2012
3: Avanish Lakshman & Prashant Malik, Cassandra - A Decentralized Structured Storage System, ACM SIGOPS Operating Systems Review archiveVolume 44 Issue 2, 2010-04
4: Edlich, Friedland, Hampe, NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken, Kapitel 3 "Wide Column Stores", Carl Hanser Verlag GmbH & CO. KG, 2010-10
5: Dietrich Featherston, Cassandra: Principles and Application, University of Illinois, 2010-07
6: HBase, -, http://de.wikipedia.org/wiki/Hadoop#HBase, Letzter Aufruf: 01.04.2012
7: Hadoop Distributed File System (HDFS), -, http://de.wikipedia.org/wiki/Hadoop#Hadoop_Distributed_File_System_.28HDFS.29, Letzter Aufruf: 01.04.2012
8: Kannan Muthukkaruppan, The Underlying Technology of Messages, 2010-11, http://www.facebook.com/note.php?note_id=454991608919, Letzter Aufruf: 01.04.2012
9: Kannan Muthukkanuppan, HBase@Facebook - The Technology Behind Messages (and more…) , 2011-03, http://qconlondon.com/dl/qcon-london-2011/slides/KannanMuthukkaruppan_HBaseFacebook.pdf, Letzter Aufruf: 01.04.2012
10: Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, Rodrigo Schmidt, Amitanand Aiyer, Apache Hadoop Goes Realtime at Facebook, SIGMOD '11 Proceedings of the 2011 international conference on Management of data, 2011-
11: MySQL, -, http://de.wikipedia.org/wiki/Mysql, Letzter Aufruf: 01.04.2012
12: Running MySQL at Scale Tech Talk, 2010-11, http://www.facebook.com/photo.php?v=695491248045, Letzter Aufruf: 01.04.2012
13: Derrick Harris, Facebook trapped in MySQL ‘fate worse than death’, 2011-07, http://gigaom.com/cloud/facebook-trapped-in-mysql-fate-worse-than-death/, Letzter Aufruf: 01.04.2012