Projektbericht

Hamaube

Kai Martinen, Fabian Kohler, ...

June 12, 2018

Universität Hamburg

Department of Computer Science

Chair of Distributed Systems and Information Systems

Betreut durch Steffen Friedrich

Abstract

Abstract in English

Kurzfassung

Kurzfassung auf Deutsch

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Introduction.
You can reference the only entry in the .bib file like this: [**?**]

1.1 Initial goal and contributions

1.2 Thesis outline

Chapter 2

Preliminaries

2.1 Topic 1

2.1.1 Subtopic1

2.1.2 Subtopic2

2.2 Topic 2

2.2.1 Subtopic1

Chapter 3

Conclusion

Write here you conclusions

3.1 Future work

Appendix A

Glossary

Just comment `\input{AppendixA-Glossary.tex}` in Masterthesis.tex if you don't need it!

Symbols

$	US. dollars.

A

A	Meaning of A.

B

C

D

E

F

G

H

I

J

M

N

P

Q

R

S

T

U

V

W

X

Appendix B

Appendix C

Cassandra ist eine NoSql Datenbank, die nach dem Wide-Column Model konzipiert ist. Cassandra vereint verschieden Eigenschaften von Googles BigTable [1] und Amazons Dynamo [2]. Sie wurde 2010 von Facebook entwickelt, um das Inbox Search Problem zu lösen [3]. Es ist eine verteilte Datenbank, die sich unter dem CAP-Theoram als AP charakterisieren lässt, wobei man das Level an Konsistenz manuell einstellen kann.

B.1 Daten Model

Das Daten Modell von Cassandra entspricht dem von BigTable, also dem Wide-Column Modell. Dies beruht im Wesentlichen, wie alle Arten an NoSql Datenbank Modellen auf Key-Value-Stores. Dabei wird für einen Primary Key jeweils eine Reihe mit Column-Families definiert. Die Column-Families bestehen wiederum aus einem Key, der sie beschreibt, und einem Value der dann den Wert an gibt. Der Wert kann allerdings wieder eine Menge an Column-Families sein, wodurch es möglich ist Column-Families beliebig zu verschachteln. Es wird bei der Initialisierung angegeben welchen Typ der Wert hat, Verschachtlung wird über benutzerdefinierte Typen erzeugt. Column-Families, die wiederum Column-Families beinhalten, bezeichnet man als Super-Column-Families. Bei Cassandra werden bei der Initialisierung einer Tabelle, die auch nichts anderes ist als eine Super-Column-Family, alle möglichen Column-Families angegeben. Sie definiert darüber hinaus den Primary Key über den auf die Werte der Column-Families zugegriffen wer-

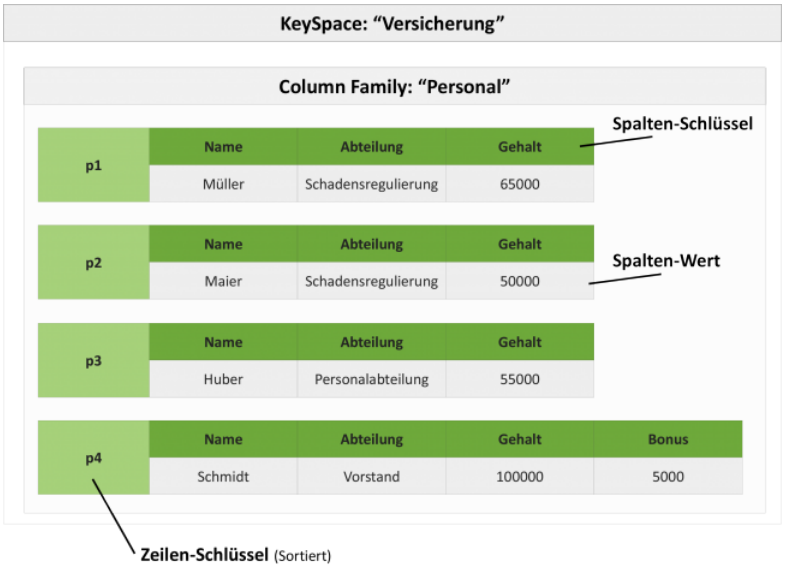


Figure B.1: Beispiel Daten Modell

den können. Im Unterschied zu herkömmlichen SQL Tabellen ist es aber möglich Werte für diese zu unterschlagen, wie in Abbildung B.1 für p1 - p3 dargestellt.
Ein KeySpace stellt die oberste Schicht des Datemodells da. Für den KeySpace werden bei der Initialisierung eine Replikationsstrategie und eine Anzahl Replikas angegeben, die zu erstellen sind. Diese gelten dann für alle Tabellen, die unter dem KeySpace erstellt werden.

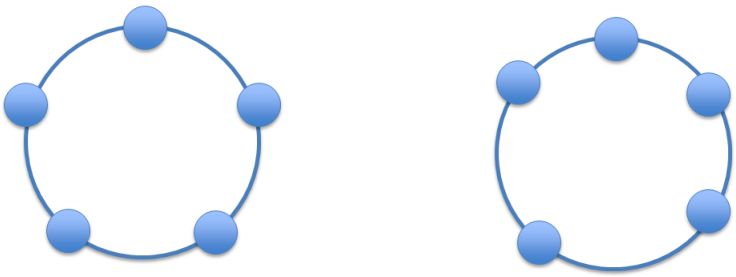
B.2 System

Implementiert ist Cassandra in Java. Darauf aufbauend sind auch die Basis Java Typen verfügbar. Die Daten werden von Cassandra redundant auf verschiedene Instanzen verteilt. Systemnachrichten werden dabei über UDP verschickt, Anwendungsnachrichten, also Nachrichten die mit den Daten zu tun haben, per TCP um den Verlust von Nachrichten zu vermeiden. Bei den Systemnachrichten ist dies zu verkraften.

B.2.1 Partitionierung

Die Partitionierung orientiert sich an der von Dynamo [2]. Cassandra benutzt genauso wie Dynamo Consistent-Hashing, um Daten auf die ver-

schiedenen Instanzen zu verteilen. Dabei erhalten die verschiedenen Instanzen einen Wert, der sie uniform über einen vordefinierten Wertebereich verteilt, wie in Abbildung B.2a abgebildet. Consistent-Hashing macht aus dem Wertebereich einen Ring, über den dann die Daten auf die Instanzen wie folgt verteilt werden. Aus einem Datum wird über die Hashfunktion ein Hash-wert berechnet. Das Datum wird dann auf der Instanz abgespeichert, deren Wert auf dem Ring aufsteigend als nächstes kommt. Dieses Verfahren kann zu einer ungleichen Verteilung der Daten auf die Instanzen führen, sodass dadurch die Performance des Systems ineffizient wird. Cassandra löst diese Problem anders als Dynamo dadurch, dass die Werte der Instanzen an die Verteilung der Daten angepasst werden, wie in Abbildung B.2b dargestellt. So sind zwar einige Instanzen für einen größeren Wertebereich zuständig, andererseits kommen in diesem größeren Wertebereich weniger Daten vor, sodass die Daten uniform auf die Instanzen verteilt werden. Wird der Datensatz zu groß, skaliert Cassandra, indem eine Instanz im Consistent-Hashing Ring zwischen den Knoten mit den Meisten Daten eingefügt wird. Danach werden die Bereiche wieder so angepasst, dass alle Instanzen ungefähr gleich belastet sind.



(a) Consistent-Hashing uniform distributed instances

(b) Consistent-Hashing adapted distribution of instances

Figure B.2: Consistent-Hashing Ring

B.2.2 Replikation

Die Art der Replikation in Cassandra ist vom Benutzer konfigurierbar. Die Anzahl der Replikas und die Replikationsstrategie wird durch den KeySpace festgelegt. Dabei kann man zwischen SimpleStrategy und NetworkTopologyStrategy auswählen. Die SimpleStrategy repliziert ohne auf die Netzw-

erkstruktur einzugehen. Somit beugt sie weniger stark potentiellem Datenverlust vor und sollte daher nur für Test-Zwecke benutzt werden . Sei N die Anzahl Replikas, werden die Daten immer auf die N-1 Nachfolgeknoten repliziert. Bei der NetworkTopologyStrategy wird die Hierarchie von Datacentern und drin enthalten Racs bei der Verteilung betrachtet wird. Somit wird diese Strategie auch für das Deployment empfohlen. Innerhalb dieser Strategie kann man sich wiederum zwischen Rack Aware und Datacenter Aware entscheiden. Dabei werden die Replikas entweder auf verschiedene Racks oder Datacenter verteilt, um die höchst mögliche Datensicherheit zu gewährleisten. Diese Strategien beziehen sich auf den Koordinator, also die Hauptinstanz einer Partition von Daten, da dieser für die Replikation zuständig ist. Bei der Bestimmung eines Koordinators wird Zookeeper verwendet. Dadurch sind alle Netzwerkänderungen und -konfigurationen persistent gespeichert, da Zookeeper die Konfigurationen jedes Knotens automatisch persistent speichert. Zur Kommunikation werden bei Zookeeper Gossip Algorithmen verwendet.

B.2.3 Persistenz

Persistenz erreicht Cassandra über eine ähnliches System wie BigTable [1]. Zunächst gibt es die MemTable, die im Hauptspeicher gehalten wird und als Cache fungiert. Sie besitzt eine Konfiguration einer Schranke ab der die MemTable auf die Platte persistiert wird. Auf der Platte gibt es die SSTable, Bloom Filter, index file, compression file und statistics file. Die Daten werden in eine SSTable geschrieben, also eine eigene Datei geschrieben, wenn sie noch nicht vorhanden sind. Wenn dies nicht der Fall ist, wird die betreffende Teil einer SSTable in die Memtable geladen, alle Operationen ausgeführt und die Daten wieder zurück auf die Platte geschrieben. Der Bloom Filter verhindert unnötige Lookups in nicht relevante SSTables. Der Index beschleunigt den Lookup innerhalb einer SSTable. Damit man nicht viele kleine SSTable-Dateien hat werden zwei SSTable, durch einen merge-Prozess immer dann zusammengefasst, wenn eine mindestens halb so groß ist wie die andere. Vorhanden SSTable files können zusätzlich noch komprimiert werden.

Figure B.3: CQL Mapping

B.3 CQL

Mit CQL (Cassandra Query Language) gibt es eine auf Cassandra zugeschnittene SQL-ähnliche Abfragesprache, die es den Anwendern konventioneller Datenbanken deutlich leichter macht mit Cassandra zu arbeiten. Dabei ist es wichtig zu Wissen dass CQL bei weitem nicht so ausdrucksstark ist wie SQL. Das liegt daran, dass CQL im wesentlich eine abstrakte API für das Cassandra Datenmodell darstellt. In CQL sind normale Datenbank Typen wie int, text, etc. möglich, allerdings kann man auch von Collections wie List Set und Map Gebrauch machen, da es dafür direkte Java Typen gibt. Des Weiteren ist es möglich eigene Typen zu definieren, wie schon in Abschnitt B.1 beschrieben.
Tabellen und Spalten werden wie ebenso in Abschnitt B.1 beschrieben durch Column-Families dargestellt. und wie in SQL erzeugt. Dabei wird ein Primary Key benötigt der dann als Row Key fungiert. Es kann nur über diesen Row Key auf die Zeilen zugegriffen werden. Deshalb kann man in der WHERE-Klausel in Cql auch nur Elemente des Primary Key angeben. Auf diesen Elementen wird durch die Indizierung schon beim speichern in Cassandra eine Sortierung berechnet was die Anfragen deutlich perfomanter macht. Für alle anderen Spalten der Zeile wäre dies also inperformant und wird von CQL nicht gestattet. Die Spalten und Zeilen werden wie folgt auf das Cassandra Datenmodell abgebildet.

Der erste Teil des Primary Keys bildet wie man sehen kann den Row Key, die restlichen Teile werden mit ihren Werten in die Beschreibung der Column-Families integriert, so wie in Abbildung B.3 beschrieben. Somit werden alle Zeilen mit dem gleichen ersten Teil des Primary Keys in der gleichen Zeile abgespeichert. Über dieses Mapping ist es möglich eine tabellenartige Abstraktion zu erzeugen, die sich durch CQL ausdrückt.

Appendix C

Appendix C

Some text.

C.1 Einführung

Täglich kommen mehrere Petabytes an Daten von über 60 Google Anwendungen zusammen. Dafür verantwortlich sind mehr als 1000 Computer die untereinander vernetzt sind. Um diese Daten verwalten zu können wurde Bigtable ins Leben gerufen. Das Ziel der Datenbank war es in vielen Bereichen anwenden zu können. Dazu sollte es Skalierbar sein sowie eine hohe Performance und Verfügbarkeit besitzen.

C.2 Datenmodell

Bigtable ist eine verteiltes, persistentes multidimensionale sortierte Map. Diese Map ist indexiert über eine row key, columen key und einem timestamp. Jeder Wert in dieser Map ist ein Array mit Bytes. Das folgende Datenmodell soll eine Speicherung von Webseiten veranschaulichen.

Das Datenmodell besteht aus zwei Familien dem Inhalt und den Ankerpunkten. Die erste Familie beinhaltet den Inhalt der Webseite, mit drei unterschiedlichen Zeitstempeln (t3,t5,t6). Drei unterschiedliche Zeitstempeln bedeutet, dass die Website www.cnn.com in drei unterschiedlichen Versionen abgespeichert wurde. Die Anker Familie beinhaltet jeweils nur eine Version. Den Anker mit „CNN.com“ mit dem Zeitstempel t8 und dem Anker „CNN“

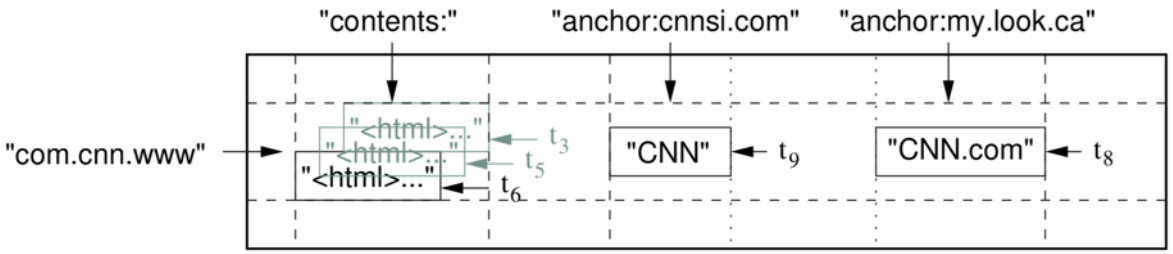


Figure C.1: Tabellen Beispiel

mit dem Zeitstempel t9.

Rows Bigtable speichert Daten in lexikopgrahischer reihenfolge und sortiert diese nach Zeilen. Eine row range beinhaltet alle gleichnamigen URL's, so dass alle mit der gleichen Domain zusammen abgespeichert werden. Das vereinfacht die Analyse und das Hosting der gleichnamigen Domains und macht dies zudem effizienter.

Column families Verschieden column keys werden in eine gemeinsame Gruppe gespeichert. Das nennt man column families. Alle Daten, welche in der gleichen Gruppe gespeichert werden sind vom gleichen typ. Bevor Daten in einer Gruppe gespeichert werden können, muss diese column family als erstes erstellt werden.

Timestamp Jede Zelle in Bigtable kann mehrere Versionen der gleichen Daten beinhalten. Dieser Versionen werden indexiert durch den Zeitstempel (timestamp). Der Zeitstempel ist bis auf die Micro Sekunde genau. Durch die „two per-column-family“ kann der Benutzer festlegen, wie viele Versionen der gleichen Daten gespeichert werden sollen. Alle weiteren Versionen werden automatisch gelöscht.

API Die API von Bigtable ermöglicht das erstellen und löschen von Tabellen und Spaltennamen, sowie das ändern von Tabellen, Cluster und Metadaten einer Spaltenfamilien. Das folgende Codebeispiel wurde in C++ geschrieben und Verändert den Inhalt der Tabelle Webtable.

Der Code verändert die Spalte „com.cnn.www“ und fügt einen neuen Anker hinzu. Im nächsten Schritt wird ein vorhandener Anker „anchor:www.abc.com“

```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

Figure C.2: Zugriffs Beispiel mit der BigTable API

gelöscht und festgeschrieben.

Building Blocks Bigtable ist auf mehreren anderen Teilen der Google-Infrastruktur aufgebaut. Zum Beispiel benutzt Bigtable das verteilte Google File System (GFS). Welches für die Speicherung von Logs und Daten verantwortlich ist. Ein Bigtable Cluster operiert typischerweise auf einem verteilten Pool von Computern. Auf diesem Pool laufen eine breite sparte von verschiedenen Anwendungen. Bigtable basiert auf einem Cluster Management System für Zeit-Planung-Jobs, Ressourcenmanagements auf geteilten Computer, agieren mit Computerfehlern und dem Anzeigen des Computer Status. Das SSTable Format stellt eine persistente, geordnete unveränderliche Abbildung von Schlüsseln zu Werten zur Verfügung, wobei sowohl Schlüssel als auch Werte willkürliche Bytefolgen sind. Operationen werden bereitgestellt, um den Wert zu suchen, der einem bestimmten Schlüssel zugeordnet ist.

Implementation Bigtable besteht aus drei Haupt Komponenten, einer Bibliothek, einem Master Server und vielen weiteren tablet servern. Die Bibliothek ist in jeden Client verlinkt, somit kann der Client auf alle Funktionen zugreifen. Der Master Server wird zufällig ausgewählt. Dieser teilt den tablet servern die tablets zu. Außerdem ist für die Verteilung der Lasten zuständig und ist für die garbage collection. Sobald eine Tabelle zu groß wird, wird diese von einem tablet server gesplittet. So wird sichergestellt, dass eine Tabelle nie größer als 100-200 MB ist.

Tablet location Um Daten zu speichern, verwendet Google bei Bigtable eine „three-level hirarchy“. Das erste Level ist eine Datei, welches auch das Chubby file genannt wird, dort wird der Speicherort des root tablets hinterlegt. Das root tablet beinhaltet alle Speicherorte aller tablets in einer METADATA Tabelle. Das spezielle an dieser Tabelle ist, dass egal wie groß sie wird, diese niemals geteilt wird. Somit wird sichergestellt, dass die „three-level hirarchy“ eingehalten wird. Die METADATA Tabellen speichern die Orte aller anderen tablets in einer Tabelle ab.

Tablet Zuweisung Jedes tablet ist zu einem Zeitpunkt immer nur einem tablet server zugeordnet. Der Master server verfolgt die lebenden tablet servern und die aktuell zugeordneten tablets zu den tablet servern inklusive aller unzugeordneten tablet servern. Beim Start einer Bigtable führt der Master Server folgende Schritte aus:

1. Wählt einen einzigartigen Master Lock in Chubby
2. Scannt die Server Verzeichnisse um die lebenden tablet server zu finden
3. Kommuniziert mit den vorhandenen tablet server um bereits zugeordnete tablets zu identifizieren
4. Master scannt METADATA Tabelle um die vorhandene Zugehörigkeiten zu lernen

Tablet serving Ein persistenter Zustand eines tablets wird in GFS gespeichert. Alle Updates werden auf „well-formed“ geprüft und anschließend in einem commit-log gespeichert. Die neusten Updates werden in eine memtable gespeichert, ältere updates werden in die SSTable geschrieben. Wenn Daten aus dem tablet server abgefragt werden muss ein merge zwischen den neuen Daten in der memtable sowie den älteren Daten in der SSTable erstellt werden.

Compaction Je mehr Daten gespeichert werden, umso größer wird die memtable. Damit diese tabelle nicht zu groß wird, gibt es ein „minor compaction“. Diese Funktion friert eine memtable ein sobald diese eine bestimmte Größe erreicht hat und erstellt eine neue memtable. Die gefrorene

memtable wird zu einer SSTable konvertiert. Je mehr Daten gespeichert werden, desto unordentlicher wird die Ansammlung von SSTable. Um die SSTable zu sortieren wird periodisch ein „merging compaction“ ausgeführt. Dies strukturiert die SSTable neu und es werden Ressourcen, durch die Löschung von Daten, freigegeben. Außerdem werden gelöschte Daten endgültig gelöscht, das ist wichtig für Services, welche sensible Daten beinhalten.

C.3 Refinments

Um die hohe Performance, Verfügbarkeit und Zuverlässigkeit beizubehalten, werden einige Verbesserungen (refinments) benötigt.

Lokale Gruppen Gruppierung erspart Zugriffszeit. Zum Beispiel bei dem Datenmodell Webseite. Die „page metadate“ und „content“ der Webseite werden in einer anderen gruppe gespeichert. So muss eine Anwendung, welche nur die Metadaten möchte, nicht durch den kompletten Inhalte einer Seite iterieren. Zudem gibt es Tuningparameter, welche bestimmten ob Daten in den Arbeitsspeicher geladen werden um die Zugriffszeit zu minimieren.

Kompression Ein Benutzer kann selbst bestimmen ob SSTable komprimiert wird und falls ja, in welchem Ausmaß. Jeder SSTable Block kann einzeln ausgewählt werden. Für die Komprimierung kommen die verfahren Bentley und McLlroy's zum Einsatz. Diese können mit 100-200Mb/s Kodiert und mit 400-1000 MB/s Enkodiert werden.

Caching für Lesezugriffe Für das Caching von Lesezugriffe gibt es zwei verfahren. Der Scan Cache (high-level), speichert key-value Paare und liefert eine SSTable zurück. Das Block Cache (lover-level) verfahren speichert SSTable Blocks, die von der GFS gelesen werden.

Bloom Filter Benutzer legt selbst fest ob ein Filter zum Einsatz kommt. Der Vorteil eines Filters liegt darin, dass wenn Daten gesucht werden, nicht jede SSTable nach den bestimmten Daten durchsucht werden muss. Ein Bloom Filter erlaubt es, nach einer bestimmten Art von row/column paaren zu fragen, ob diese in einer SSTable gespeichert sind.

Beschleunigte tablet Wiederherstellung Wenn der Master ein tablet von einem Server zu einem anderen Server verschiebt, führt der Ursprungs Server erst ein „minor compaction“ aus um die Ladezeit für den neuen tablet server zu verkürzen.

Unveränderlichkeit ausnutzen Es können nur Daten verändert, welche in der memtable stehen. Daten in der SSTable können nicht verändert werden. Das macht man sich zu nutzen indem man keine Synchronisation braucht, wenn auf die Daten zugegriffen wird. Memtable sind die einzigen Daten auf die man schreiben kann und gleichzeitig lesen. Damit es zu keinen komflikten kommt, setzt Bigtable hier auf „Copy-on-write“.

C.4 Performance Auswertung

	# of Tablet Servers			
Experiment	**1**	**50**	**250**	**500**
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

Figure C.3: Percormance Übersicht

Die Performance wird hauptsächlich durch die verwendete CPU (2ghz) begrenzt. Zudem kann man erkennen, dass bei einem tablet server der Durchsatz bei ca. 75MB/s liegt (1000 bytes * 64 KB Block size = 75 MB/s). Damit der Durchsatz bei einem Single tablet server erhöht wird, wird die die SSTable größe in der regel von 64KB auf 8kb gesenkt. Zudem wird erkannt, dass der Durchsatz nicht Linear ansteigt. Bei einer Erhöhung der tablet server von eins auf 500 liegt die Erhöhung des Durchsatzes bei gerade mal dem 300 fachen (10811 / (500 * 6250) = 350). Diese Begrenzung liegt wie bei einem tablet server an der CPU der tablet servern.

Appendix D

Appendix C

Some text.

D.1 Section 1

D.2 Section 2

Appendix E

Appendix C

Some text.

E.1 Section 1

E.2 Section 2

Appendix F

Appendix C

Some text.

F.1 Section 1

F.2 Section 2

Appendix G

Appendix D

G.1 Section 1

Bibliography

[1] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[3] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.