Hochschule RheinMain
Wintersemester 2015/16

Fachseminar Datenbank-Optimierung

Google Chubby – Ein verteilter Lockservice

Patrick Canterino

14. Februar 2016

1 Abstract

Google Chubby ist ein intern bei Google verwendeter Lockservice, der von verteilten Systemen genutzt werden kann. Die verteilten Systeme nutzen Chubby, um den Zugriff auf gemeinsam genutzte Ressourcen zu regeln, um gemeinsame Entscheidungen zu treffen und um Informationen auf einem hochverfügbaren Speicher abzulegen.

2 Einleitung

Chubby ist ein bei Google verwendetes verteiltes System. Es stellt anderen verteilten Systemen ein einfaches Dateisystem mit Sperrfunktionen zur Verfügung. Diese können es zur Synchronisierung und als Datenablage verwenden.

In dieser Ausarbeitung wird zunächst auf die Motivation hinter dem Projekt und die Design-Grundlagen von Google Chubby eingegangen. Anschließend wird der serverseitige Aufbau einer sog. Chubby-Zelle und die Möglichkeiten, die sie dem Client bietet, beschrieben. Danach wird die Kommunikation zwischen Chubby-Zelle und Client-Bibliothek erläutert und welche Sicherungsmechanismen existieren. Zum Schluss werden die Erfahrungen sowie die Nutzung innerhalb von Google beschrieben.

3 Literatur

Mike Burrows, Mitarbeiter von Google, liefert in seinem im November 2006 veröffentlichten Paper *The Chubby lock service for loosely-coupled distributed systems* sehr wichtige Informationen zu Google Chubby. Er geht detailliert auf den Aufbau, Sicherungs- und Erweiterungsmechanismen sowie auf die Erfahrungen, die man durch die Nutzung gesammelt hat, ein.

Im Internet finden sich mehrere englischsprachige Präsentationen von verschiedenen Personen, die das Paper von Burrows als Grundlage verwenden und eine Kurzfassung bieten. Diese Präsentationen bieten jedoch im Vergleich zu Burrows' Paper keine neuen Informationen.

In dieser Ausarbeitung werden die wichtigsten Informationen aus Burrows' Paper zum Aufbau von Chubby zusammengefasst.

4 Google Chubby

4.1 Grundlagen

Bevor Chubby entwickelt wurde, nutzten die verschiedenen verteilten Systeme innerhalb von Google jeweils eigens für dieses System entwickelte Methoden, um den Zugriff auf gemeinsam genutzte Ressourcen zu regeln und um gemeinsame Entscheidungen zu treffen. Zu diesen gemeinsamen Entscheidungen gehört bspw. einen Master innerhalb eines solchen Verbundes auszuwählen.

Die Motivation bei der Entwicklung von Google Chubby war, diese verschiedenen Methoden zu vereinheitlichen, ohne dass die Anwendungen dazu auf komplizierte Weise angepasst werden müssen. Bei der Entwicklung stand im Vordergrund, dieses System verlässlich sowie für eine große Zahl an Clients zu konzipieren und die Entwickler-Schnittstelle so einfach wie möglich zu halten. Ziele wie Performance standen nur an zweiter Stelle.

Die Entwickler standen vor der Wahl, Chubby als Paxos-basierte Bibliothek oder als Dienst zu entwickeln. Man entschied sich aus mehreren Gründen für einen Dienst, den die verteilten Systeme gemeinsam nutzen sollen. Zum einen ist der Aufwand, in einem bestehenden System besagten Dienst zu nutzen, geringer, als an einem Entscheidungsprotokoll wie Paxos teilzunehmen. Zum anderen können die Teilnehmer eines verteilten Systems über einen gemeinsam genutzten Dienst Informationen über ihren Zustand bereitstellen oder diesen als Quorum nutzen. Weiterhin sind die Grundlagen von Sperrverfahren für Entwickler leicht verständlich.

Chubby ist hauptsächlich für grobkörnige Sperren entworfen worden. Das sind Sperren, die zum einen große Bereiche an Ressourcen vor konkurrierendem Zugriff schützen und die zum anderen auch eine vergleichsweise lange Zeit aktiv sind. Chubbys Sperren sind zudem nur empfehlend (*advisory*), sie gelten also wie Mutexe oder Semaphoren nur für Anwendungen, die diese explizit nutzen.

4.2 Technik

4.2.1 Zelle

Grundlage eines Chubby-Systems ist die sog. *Zelle*. Eine solche Zelle besteht zumeist aus fünf Servern, die über ein schnelles Netzwerk miteinander verbunden sind. Meistens befinden sich die zu einer Zelle gehörenden Server in räumlicher Nähe zueinander, es ist jedoch auch möglich, dass diese sich in verschiedenen Ländern befinden. Aus Sicherheitsgründen werden die zu einer Zelle gehörenden Server zumindest auf unterschiedliche Racks im Serverraum verteilt.

Jede Zelle wählt mit Hilfe von Paxos einen *Master*. Wenn ein Master gewählt wurde, versprechen die restlichen Server in der Zelle (*Replicas*) dem Master, dass sie für einen als *Master Lease* bezeichneten Zeitraum keinen anderen Server zum Master ernennen. Der Master Lease wird regelmäßig erneuert, vorausgesetzt die Replicas würden den bisherigen Master erneut zum Master ernennen.

Die Server einer Zelle werden im DNS registriert. Ein Client kann den Master nun finden, indem er eine Anfrage an die Server der Zelle stellt. Replicas antworten auf eine solche Anfrage mit dem Namen des Masters.

Jeder Server in der Zelle verfügt über Kopien einer Datenbank. Hier wird eine einfache Schlüssel-Werte-Datenbank mit Write-Ahead-Logging und Snapshot-Funktionen genutzt. Lese- und Schreibvorgänge auf die Datenbank werden ausschließlich durch den Master ausgeführt, Veränderungen an der Datenbank werden mit Hilfe von Paxos an die Replicas übertragen. Diese Datenbank wird regelmäßig durch den Master auf einen Server in einem anderen Gebäude gesichert.

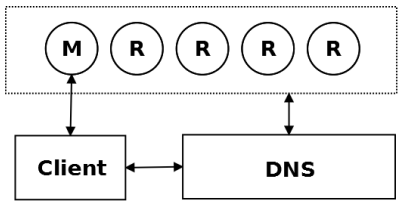

Abbildung 1: Aufbau einer Chubby-Zelle, eigene Darstellung nach [Bur06]

4.2.2 Dateisystem

Chubby stellt ein einfaches Dateisystem zur Verfügung, das einem normalen Unix-Dateisystem ähnelt. Dieses Design wurde zum einen gewählt, da es für Entwickler leicht verständlich ist. Zum anderen kann auf diese Weise ein Zugriff durch andere Systeme wie das Google File System einfacher realisiert werden, da es vom Grundprinzip her Gemeinsamkeiten gibt.

Das von Chubby bereitgestellte Dateisystem besteht aus einem Baum von *Knoten*, also Dateien und Verzeichnissen. Ein Dateipfad sieht üblicherweise so aus (Beispiel entnommen aus [Bur06]):

```
/ls/foo/wombat/pouch
```

Dieser Pfad ist folgendermaßen aufgebaut:

- Der erste Teil `ls` ist ein gemeinsamer Präfix und steht für ***L**ock **S**ervice*.
- Der zweite Teil (in diesem Fall `foo`) ist der Name der Chubby-Zelle. Hier steht auch eine spezielle Chubby-Zelle namens `local` zur Verfügung, die sich oftmals im selben Gebäude befindet wie der Client, der eine Anfrage an diese Zelle stellt. Weiterhin existiert eine Zelle namens `global`, auf die weltweit zugegriffen werden kann.
- Der Rest (in diesem Fall `wombat/pouch`) wird von der jeweiligen Zelle interpretiert.

Das Dateisystem ist einfacher als die normalen Unix-Dateisysteme. Es werden bspw. keine Verschiebe-Operationen zur Verfügung gestellt, da dies bedeuten würde, Dateien aus einer Chubby-Zelle in eine andere zu verschieben. Weiterhin werden bestimmte Meta-Daten nicht zur Verfügung gestellt, um das Caching zu vereinfachen oder weil diese Daten oft nicht relevant sind. Zu diesen Daten gehört das Änderungsdatum eines Verzeichnisses und wann zuletzt auf eine Datei zugegriffen wurde. Außerdem gibt es keine symbolischen oder harten Links. Des weiteren gibt es in neueren Chubby-Versionen ein Limit für die Dateigröße von 256 KB.

Die Knoten im Dateisystem können permanent oder kurzlebig sein. Kurzlebige Dateien werden gelöscht, wenn sie von keinem Client mehr geöffnet sind. Aus diesem Grund werden sie häufig von Clients genutzt, um zu melden, dass sie aktiv sind. Kurzlebige Verzeichnisse werden gelöscht, wenn sie leer sind.

Zu jedem Knoten gibt es Meta-Daten. Dazu gehören ACLs (**A**ccess **C**ontrol **L**ists), verschiedene Zähler, die eindeutige Nummern für den Knoten, den Inhalt, die Sperren und die ACLs bereitstellen sowie eine 64-Bit-Prüfsumme für den Inhalt von Dateien.

Wenn ein Knoten geöffnet wird, wird ein *Handle* erzeugt, über das der weitere Zugriff ermöglicht wird. Jedes Handle enthält eine Prüfsumme, die verhindern soll, dass Clients Handles fälschen. Weiterhin enthalten sind eine Nummer, die das Handle einem Master zuordnet und den Modus, in dem das Handle geöffnet wurde. Diese Informationen sind nötig, falls das Handle an einen anderen Master übertragen werden muss.

Es ist möglich, das Dateisystem einer Zelle in *Partitionen* aufzuteilen, die dann wiederum aus einem Master und Replicas bestehen. Die Verzeichnisse im Dateisystem werden mit Hilfe von Hash-Funktionen einer Partition zugeteilt. Dies ist ein weiterer Grund, weshalb keine Verschiebe-Operationen zur Verfügung gestellt werden, da man sonst Daten aus einer Partition herausschieben könnte, was

nicht gewünscht ist. Durch Partitionierung ließe sich die Last in der Zelle besser verteilen. Von dieser Möglichkeit wurde bis 2006 jedoch kein Gebrauch gemacht.

4.2.3 Sperren

Jeder Knoten (also Dateien **und** Verzeichnisse) in einem Chubby-Dateisystem kann mit einer exklusiven Sperre für Schreibzugriffe oder einer Sperre für Lesezugriffe, bei der mehrere Clients lesen dürfen, belegt werden. Wie bereits erwähnt, sind die Sperren empfehlend und nicht verpflichtend (*mandatory*). So ist es bspw. einfacher, Dateien für Debug-Zwecke zu öffnen, da der Zugriff durch einfaches Ignorieren der Sperre immer noch möglich ist. Das Setzen einer Sperre verlangt Schreibzugriff auf den Knoten.

Jeder Client hat die Möglichkeit, einen sog. *Sequencer* von der Chubby-Zelle zu erhalten, nachdem eine Sperre gesetzt wurde. Dieser Sequencer ermöglicht, den Status der Sperre zu prüfen. Ein Chubby-Client kann einen Sequencer an einen von ihm genutzten anderen Dienst senden, der diesen dann bei der Chubby-Zelle gegenprüfen kann.

Wird eine Sperre regulär freigegeben, kann der Knoten nun sofort von anderen Clients gesperrt werden. Stellt Chubby jedoch fest, dass der Client nicht mehr erreichbar ist, wird die Sperre für einen vorher vom Client festgelegten Zeitraum gehalten (*Lock Delay*), der Standardwert ist eine Minute. Auf diese Weise kann eine evtl. später noch eintreffende Nachricht des betroffenen Clients noch unter dem Schutz der Sperre ausgeführt werden.

4.2.4 Ereignisse

Chubby sendet unter bestimmten Bedingungen Ereignisse (*Events*) an den Client, die dann durch diesen entsprechend bearbeitet werden können. Die Ereignisse betreffen die Handles. Ein Client kann so bspw. darauf reagieren, dass eine Datei verändert, gelöscht oder gesperrt wurde. Weiterhin kann so darauf reagiert werden, dass ein anderer Client versucht, eine Datei zu sperren, ein Handle ungültig geworden ist oder der Master ausgefallen ist.

4.2.5 Spiegelung

Es ist möglich, Dateien von einer Chubby-Zelle in eine andere zu spiegeln. Hierzu wird der Ereignis-Mechanismus genutzt. Dies und die Tatsache, dass Chubby-Dateien sehr klein sind, ermöglicht, dass Änderungen in weniger als einer Sekunde in einer anderen Chubby-Zelle ankommen.

Es existiert eine spezielle Chubby-Zelle namens `global`, deren Server weltweit verteilt sind. Diese Zelle enthält ein spezielles Verzeichnis namens `master`, das in andere Chubby-Zellen unter dem Namen `slave` gespiegelt wird. So kann ein Austausch von Dateien zwischen Zellen realisiert werden.

4.2.6 Client-Bibliothek

Die Chubby-Client-Bibliothek ist wie auch der Server in C++ geschrieben. Da es aufgrund des komplexen Protokolls zu aufwändig wäre, die Bibliothek in Java zu übersetzen, existieren Proxy-Server, welche die C++-Bibliothek nutzen und eine RPC-API für Java bereitstellen.

Die Client-Bibliothek stellt ein Dateisystem-Interface zur Verfügung. Durch das Öffnen einer Datei wird ein Handle erzeugt, mit dessen Hilfe die weiteren Datei- und Verzeichnisoperationen durchgeführt werden. Dazu gehören zum einen Standard-Dateioperationen wie das Lesen und Schreiben des Dateiinhaltes, das Verändern von Zugriffsrechten, sowie das Löschen und das Sperren von Dateien. Wenn eine Chubby-Datei gelesen oder geschrieben wird, betrifft dies immer die komplette Datei, es ist nicht möglich, eine Datei blockweise zu lesen oder zu schreiben.

Ein Master innerhalb eines Clusters kann mit Hilfe der Client-Bibliothek folgendermaßen gewählt und gefunden werden:

1. Alle Mitglieder des Clusters versuchen, eine Datei im Chubby-Dateisystem zu öffnen und auf diese Datei eine exklusive Sperre zu setzen.
2. Derjenige, der die Sperre setzen konnte, wird zum Master und schreibt in diese Datei, wie er erreicht werden kann.
3. Die anderen Mitglieder des Clusters oder Clients dieses Clusters können den Master nun finden, indem sie diese Datei lesen.
4. Der Master kann ggf. einen Sequencer erzeugen, um bei der Kommunikation mit anderen Systemen sicherzustellen, dass er noch der Master ist.

4.2.7 Caching

Chubby-Clients halten Dateiinhalte und Meta-Daten lokal in einem Write-Through-Cache mit Lease-Zeit vor.

Wenn der Chubby-Server eine Anfrage erhält, eine Datei zu verändern, sendet er an alle ihm bekannten Clients eine Nachricht, dass sie die entsprechenden Daten aus ihrem Cache entfernen sollen. Erst wenn alle Clients ihren Cache geleert haben (sei es durch Bestätigung vom Client oder durch Ablauf des Lease) wird die Datei verändert. Während dieser Phase kann eine Datei nicht erneut gecached werden.

Der Cache wird also nicht laufend durch den Client aktualisiert, sondern erst, wenn dieser durch den Server für ungültig erklärt wurde.

4.2.8 Sessions

Eine Chubby-Session wird begonnen, indem ein Client den Master einer Zelle kontaktiert. Sie wird durch *KeepAlives* am Leben gehalten und entweder explizit durch den Client beendet oder implizit, wenn für eine gewisse Zeit keine offenen Handles existieren und keine Aufrufe an den Server gesendet werden.

Jede Session verfügt über eine *Lease*-Zeit. Der Master garantiert, dass die Session auf jeden Fall für diese Zeit bestehen wird. Die Lease-Zeit wird unter folgenden Bedingungen verlängert:

1. Der Master fällt aus, seine Rolle wird durch einen anderen Server der Zelle übernommen (*Fail-over*).
2. Der Master erhält einen KeepAlive vom Client, dieser wird erst bearbeitet, wenn der bisherige Lease fast abgelaufen ist. Nachdem dieser bearbeitet wurde, sendet der Client sofort einen neuen KeepAlive. KeepAlive-Antworten werden auch genutzt, um Ereignisse zum Client zu senden und um den Cache für ungültig zu erklären (dabei werden KeepAlive-Antworten auch früher gesendet). Eine Session kann nicht aufrecht erhalten werden, wenn der Client nicht darauf reagiert, dass der Cache für ungültig erklärt wurde.

Jeder Client verfügt über einen eigenen Lease-Timeout, der jedoch nur eine vorsichtige Annäherung an den Timeout des Masters ist. Wenn dieser Client-Timeout abgelaufen ist, schaltet der Client seinen Cache ab und wechselt in den sog. *Jeopardy*-Modus. Der Client blockiert nun alle API-Aufrufe, wartet einen kurzen Zeitraum (*Grace Period*) und versucht währenddessen weiterhin, KeepAlives zum Server zu senden. Wenn ein KeepAlive nun korrekt beantwortet wurde, wird der Cache erneuert (*Safe*-Ereignis). Ansonsten gilt die Session als abgelaufen (*Expired*-Ereignis) und alle weiteren Operationen auf den zur Session gehörenden Handles schlagen fehl.

Es ist möglich, einen Proxy zwischen Chubby-Client und Zelle zu schalten. Er kann sich um die Behandlung von KeepAlive-Paketen kümmern und Lese-Anfragen cachen.

4.2.9 Master-Failover

Wenn der Master einer Chubby-Zelle ausfällt, so wird dies spätestens nach Ablauf des Master Lease von den restlichen Server der Zelle bemerkt. Diese reagieren darauf, indem sie mit Hilfe von Paxos einen neuen Master wählen.

Der neue Master hat vom vorherigen Master Informationen über den Zustand der Datenbank erhalten, weshalb er weiß, welche Dateien und Sperren vorhanden sind. Sessions und Datei-Handles werden jedoch im Arbeitsspeicher verwaltet und gehen bei einem Ausfall des Masters verloren. Der neue Master versucht nun, mit Hilfe der Client-Anfragen diese Informationen wiederherzustellen.

Ein Client bemerkt den Ausfall eines Masters dadurch, dass er keine Antworten auf KeepAlive-Pakete erhält. Er wechselt deswegen in den Jeopardy-Modus, sucht über DNS den neuen Master und kontaktiert ihn mit KeepAlive-Paketen. Der neue Master lehnt diese nun ggf. ab, weil die interne Master-Nummer in den Paketen nicht korrekt ist. Aber sobald passende KeepAlive-Pakete vom Client beim Master ankommen, wird die Lease-Zeit verlängert. Wenn alles gut gegangen ist, bekommt die Anwendung, die den Chubby-Client nutzt, vom Master-Ausfall nur mit, dass die Anfragen etwas länger gedauert haben.

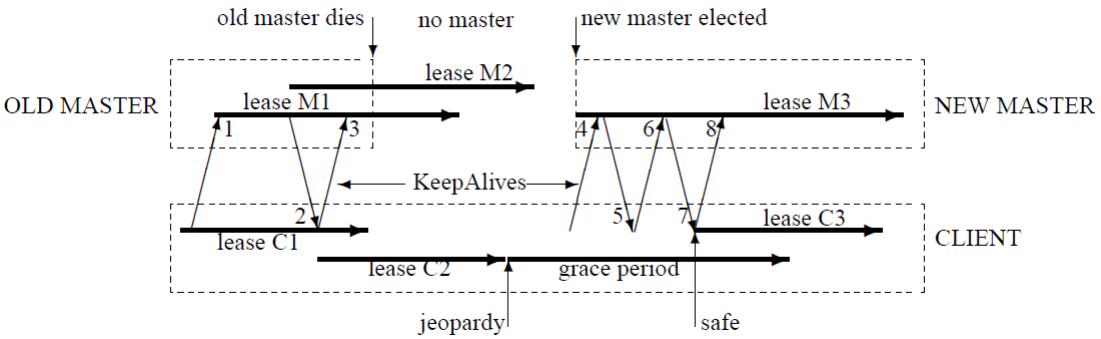

Abbildung 2: Ablauf eines Master-Failover, entnommen aus [Bur06]

4.3 Verwendung bei Google

Entwickler-Teams innerhalb von Google können für ihre Projekte eigene Chubby-Zellen aufsetzen oder gemeinsame Zellen mitnutzen. Bei Nutzung von gemeinsamen Zellen untersuchen die Chubby-Entwickler die Art und Weise, wie Chubby innerhalb eines neuen Projektes genutzt werden soll. Man möchte so vermeiden, dass Chubby für Dinge genutzt wird, für die es nicht gedacht ist.

Chubby wird bei Google auf verschiedene Weisen genutzt:

- Das Google File System (GFS) nutzt Chubby, um einen Master-Server zu ernennen, um Meta-Daten an einem hochverfügbaren Ort abzuspeichern und um die Wurzel für die verteilten Datenstrukturen zu erzeugen.
- Google Bigtable verwendet Chubby auf ähnliche Weise wie GFS. Weiterhin nutzt es Chubby, um eine Liste der vom Master kontrollierten Server zu verwalten und um den Clients den Namen des Masters bekannt zu machen. Hierbei wird auch der Spiegelungsmechanismus von Chubby genutzt, um diese Informationen weltweit verfügbar zu machen.
- Google MapReduce nutzt Chubby als Rendezvous-Mechanismus.
- Viele Dienste innerhalb von Google nutzen Chubby, um kleine Konfigurationsdateien zentral abzulegen und um diese über die `global`-Zelle weltweit zu verteilen.
- Kurzlebige Dateien werden genutzt, um ein Monitoring von diversen Systemen zu ermöglichen.
- Chubby wird jedoch hauptsächlich als DNS-Ersatz genutzt. DNS hat das Problem, dass gecachte DNS-Einträge nach einer bestimmten Zeit erneuert werden müssen. Dies kann bei einer hohen

Zahl an Prozessen zu vielen DNS-Anfragen und damit zu einer möglichen Überlastung der Server führen. Die Ungültigkeit von gecachten Chubby-Daten wird im Gegensatz dazu explizit durch die Zelle bekannt gegeben. Dies ermöglicht, dass wenige KeepAlive-Daten eines Chubby-Clients sich auf viele gecachte Daten beziehen.

- Es gibt Server, die das Chubby-Protokoll in andere Protokolle umwandeln. Zu diesen Protokollen zählt DNS.

4.4 Erfahrungen

Google hat bei der Nutzung von Chubby diverse Erfahrungen gesammelt. Diese beziehen sich auf die Zeit, in der Burrows' Paper veröffentlicht wurde:

- Ein Großteil der Dateien (90 %) auf einer Chubby-Zelle ist unter einem KB groß. 60 % der Dateien werden für Namensdienste genutzt. Lesesperren sind sehr selten, Schreibsperren häufiger.
- KeepAlive-Pakete machen 93 % des Datenverkehrs zwischen einer Chubby-Zelle und einem Client aus.
- Eine Überlastung einer Zelle tritt oft bei sehr vielen Sessions (über 90.000), bei vielen Lesezugriffen (über eine Millionen gleichzeitig) oder Fehlern in der Client-Bibliothek (Cache deaktiviert) auf. Dies führt oftmals zu Client-Timeouts.
- Bei Überlastung ist es oftmals sinnvoll, die Lease-Zeit von 12 auf 60 Sekunden zu erhöhen. Dies führt zu einer geringeren Anzahl an KeepAlive-Paketen, die der Zellen-Master bearbeiten muss. Generell zeigte sich, dass sich durch Reduzierung des Datenverkehrs die Performance einer Zelle stark verbesserte.
- Man will die Performance mit Hilfe von Proxy-Servern und Partitionierung verbessern.
- Chubby-Clients bekommen in den meisten Fällen so gut wie nichts von einem kurzen Ausfall mit, sie merken dies höchstens durch eine Verzögerung.
- Ein Datenverlust in einer Chubby-Zelle durch Hardware-Probleme ist zu diesem Zeitpunkt noch nie aufgetreten. Bis zum November 2006 gab es nur sechs Fälle von Datenverlust, diese traten durch Probleme im Datenbank-Code oder durch Bedienfehler seitens der Administratoren auf.
- Bei einem Master-Failover sind oft Sessions, in denen Dateien nur gelesen werden sollten, verloren gegangen. Sessions wurden erst in der Datenbank gespeichert, wenn eine Datei verändert, eine Sperre angefordert oder eine flüchtige Datei erstellt wurde. Sessions werden seitdem nicht mehr in der Datenbank gespeichert, sondern nur im Arbeitsspeicher verwaltet. Bei einem Failover werden diese wiederhergestellt, sobald eine neue Anfrage für diese ankommt. Dies führt dazu, dass der neue Master bei einem Failover die Sessions nun absichtlich ablaufen lässt.
- Entwickler lassen ihre Anwendungen bei einem Master-Failover von Chubby oftmals abstürzen, obwohl dies nicht nötig wäre. Weiterhin zeigte sich, dass Entwickler oftmals annehmen, dass eine Zelle immer verfügbar ist und kurze Auszeiten nicht einplanen. Die Chubby-Client-Bibliothek wurde angepasst, sodass kurze Ausfälle einer Zelle kompensiert werden.
- Es gab Fälle, in denen Chubby von Entwicklern falsch genutzt wurde.
  Chubby ist bspw. nicht dafür geeignet, große Dateien zu speichern, da diese nur als ganzes gelesen oder geschrieben werden, was den Master stark belastet. Die maximale Dateigröße wurde deswegen auf 256 KB begrenzt.
  Außerdem gab es Versuche, Chubby für Publisher-Subscriber-Anwendungen zu nutzen, für die es jedoch aufgrund der eher auf Verlässlichkeit ausgelegten Struktur nicht gedacht ist. Die Chubby-Entwickler beraten seitdem andere Entwickler, ob Chubby für ihre Zwecke geeignet ist.
  Dann gab es Fälle, in denen endlos angefragt wurde, ob eine Datei vorhanden ist. Man hat dies zum Anlass genommen, das wiederholte Öffnen von Dateien ressourcenschonender zu machen.

4.5 Vergleich mit anderen Diensten

Im Folgenden wird Chubby mit verschiedenen anderen Diensten verglichen. Sofern der Vergleich nicht aus Burrows' Paper stammt, wird eine entsprechende Quelle angegeben.

- Das Betriebssystem VMS stellt ebenfalls einen Lockservice bereit.
- Im Gegensatz zum verteilten Dateisystem AFS ist Chubby nicht auf Performance und große Dateien ausgelegt. Chubby konzentriert sich auf Verfügbarkeit.
- Das Netzwerk-Dateisystem NFS erlaubt in der Version 3 kein atomares Ändern von Dateien. Nachdem eine Datei in NFS geändert wurden, verfügen andere NFS-Clients noch über alte Informationen zum Dateiinhalt. Erst nachdem die Clients ihren Cache erneuert haben, werden die neuen Daten sichtbar. Chubby hingegen ermöglicht das Ändern einer Datei erst, wenn sichergestellt ist, dass diese aus allen Caches entfernt wurde. [Kro14]
- Der Boxwood-Lockservice von Microsoft stellt kein Dateisystem bereit, die Sperren werden hier über Byte-Sequenzen angesprochen. Boxwood hat auch andere Standardwerte bei der Zahl der Server eines Clusters und bei Timeout-Zeiten. Weiterhin sind Sperren hauptsächlich für Aufgaben innerhalb von Boxwood gedacht.
- Apache ZooKeeper ist ein Open-Source-Projekt und weist sehr große Ähnlichkeiten mit Chubby auf. Es stellt ein Dateisystem mit Sperrfunktionalität bereit. Ein ZooKeeper-Cluster verfügt über einen Master, mit dem die Clients kommunizieren können. [Fou12][Gri10]

5 Zusammenfassung

Google Chubby ist ein verteilter Lockservice, über den andere verteilte Systeme innerhalb von Google ihre Arbeit koordinieren können. Es dient hauptsächlich der Ermittlung eines Masters innerhalb dieser verteilten Systeme und als Namensdienst. Chubby ist nicht auf Performance, sondern auf Verfügbarkeit ausgelegt. Es nutzt Paxos, um einen Master innerhalb einer Zelle zu finden und um Änderungen an der Datenbank zu replizieren.

Chubby stellt seinen Clients ein Dateisystem zur Verfügung, in dem kleine Datenmengen gespeichert und Sperren realisiert werden können. Clients cachen die Daten aus dem Dateisystem, wobei der Cache explizit durch die Zelle für ungültig erklärt wird.

6 Quellen

[Bur06] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. http://research.google.com/archive/chubby-osdi06.pdf, 2006.

[Du12] Peter Du. Back to the Real World: Chubby and Bigtable. https://www.cs.columbia.edu/ du/ds/assets/lectures/lecture19.pdf, 2012.

[Fou12] The Apache Software Foundation. Apache ZooKeeper. https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index, 2012.

[Gri10] Ilya Grigorik. Distributed Coordination with Zookeeper. https://www.igvita.com/2010/04/30/distributed-coordination-with-zookeeper/, 2010.

[Jac14] Romain Jacotin. Google Chubby lock service. http://de.slideshare.net/romain_jacotin/the-google-chubby-lock-service-for-looselycoupled-distributed-systems, 2014.

[Kro14] Reinhold Kroeger. Vorlesung Verteilte Systeme, Kap. 9: Verteilte Dateisysteme. https://wwwvs.cs.hs-rm.de/lehre/material/intern/vm14ws/folien/9_VerteilteDateisysteme.pdf, 2014.