2020

Dezentralisierung von Datenspeicherung

FELIX DUMBECK

BETREUUNG: FRANK T H

MARTIN-NIEMÖLLER-SCHULE-WIESBADEN

1. Abstract

Das Projekt bietet die Möglichkeit Daten dezentralisiert, also auf Nutzergeräte verteilt, garantiert sicher, anonym und redundant zu speichern.

Dies soll geschehen, ohne eine zu große Belastung für die Rechenleistung, Internetverbindung und den Speicherplatz der Nutzer zu bedeuten.

Dafür schließen sich die Nutzer zu Clustern zusammen, die sich wiederum in einer Liste befinden.

Inhaltsverzeichnis

2. Einleitung

2.1 Problemstellung

Wie können Daten dezentralisiert, garantiert sicher, anonym und redundant abgespeichert werden, ohne dabei in *eine* Instanz vertrauen zu müssen?

Als einzige Lösung, die all diese Probleme behebt, bietet sich ein dezentralisiertes Netzwerk an, welches nicht von einer Firma besessen und kontrolliert wird, sondern als P2P-Netzwerk (Peer-to-Peer) funktioniert, also eine Verbindung aus vielen, ungefähr gleichwertigen Nutzern, die gemeinsam die Funktion von Servern übernehmen.

Es stellt sich heraus, dass „DHT's" (Distributed-Hash-Tables) wie beispielsweise „Chord" bereits 2001 von Wissenschaftlern vom MIT theoretisch, also nicht als implementierte Anwendung, entwickelt wurden.[1]

Im Allgemeinen funktionieren „DHT's" so, dass ein Knoten bzw. Nutzergerät eine gewisse Anzahl an Geräten und deren IP-Adressen kennt. Wenn der User eine Datei speichern möchte, wird diese „gehasht", um eine ID zu erstellen. Anschließend wird die Datei auf dem Knoten gespeichert, dessen ID am nächsten an der der Datei ist. Der zum Hash passende Knoten wird über ein Routingmechanismus gefunden, der ein Datenpaket immer von Knoten zu Knoten weiterleitet.

Auch wenn „Chord" sehr gut skalierbar ist und im Gegensatz zu Napster vollständig dezentralisiert ist, wird das gesamte Netzwerk beim Hinzufügen von neuen Knoten oder bei Netzwerkanfragen relativ stark belastet, was sich als problematisch erweist, wenn die partizipierenden Knoten Nutzergeräte sind. Der Grund für die Belastung ist zum einen die Verwendung eines „greedy"-Algorithmus[2][3], der Verbindungen mit sehr vielen Geräten eingeht, um den Zielknoten zu erreichen. Außerdem kommt es sehr oft vor, dass ein neuer Knoten näher an einem bereits existierenden Hash ist, was bedeutet, dass viele Daten von einem Knoten zum anderen transportiert werden müssen, was erneut eine Belastung für Nutzer darstellt (siehe Abb. 1).

Außerdem ist „Chord „nicht auf mobile Geräte ausgelegt, so stellen dynamische IP-Adressen beispielsweise ein Problem dar, da ein Knoten so seine Nachbarknoten

[1] MIT, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications: https://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf (17.01.2019)
[2] Cornell University, Papillon: Greedy Routing in Rings: https://arxiv.org/abs/cs/0507034 (17.01.2019)
[3] Springer Link: F-Chord: Improved Uniform Routing on Chord: https://link.springer.com/chapter/10.1007/978-3-540-27796-5_9 (17.01.2019)

„vergessen“ könnte. Und „Chord“ macht es dem Nutzer nicht möglich, seine Daten zu löschen.

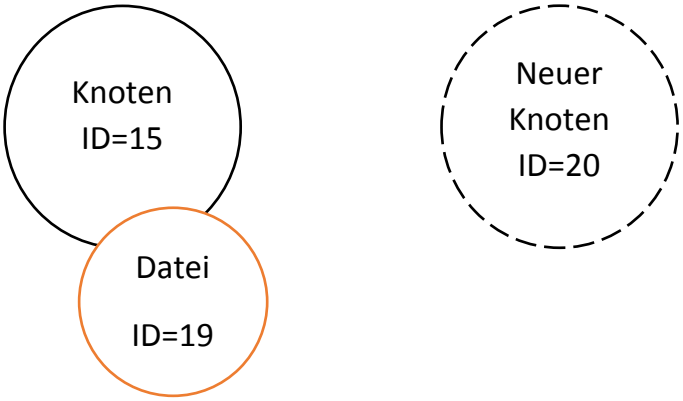

Abb. 1.1

Knoten 15 hat Datei 19 abgespeichert, da seine ID am nächsten war. Durch den Neuen Knoten ist dies allerdings nicht mehr der Fall, da 20 näher an 19 ist.

Deshalb muss die Datei nun verschoben werden, was Bandbreite und Rechenpower der beiden Knoten verbraucht, um von 1.1 auf 1.2 zu kommen.

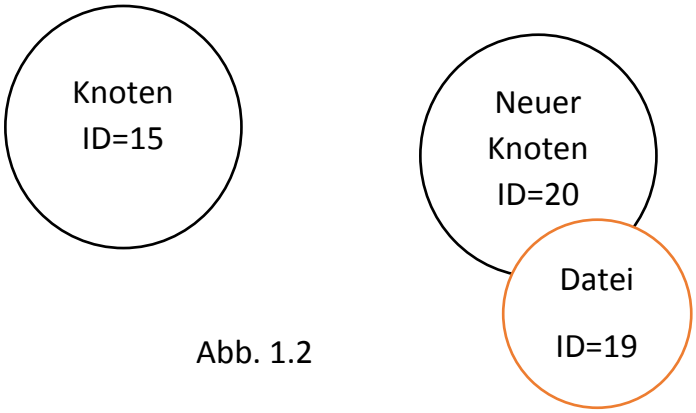

Abb. 1.2

3. Lösungsidee/ Vorgehensweise

Da das Problem aus Abbildung 1 bei jeder DHT auftritt, die signifikant weniger Knoten hat, als der Hash-Algorithmus Möglichkeiten (selbst bei SHA-1 sind es Möglichkeiten), sollte man festlegen, wo die Daten gespeichert werden ohne dies abhängig von den Daten selbst zu machen. Also erscheint es logisch, einen festen Knoten für Daten auszuwählen.

Ein einzelner Knoten würde allerdings nicht die bei der Problemstellung zum Ziel gesetzte Redundanz und damit Verlässlichkeit, dass die Daten zu jeder Zeit abrufbar sind. Allerdings haben Knoten-Cluster (mehrere miteinander verknüpfte Knoten) diese Eigenschaft und bringen mit sich, dass ein Knoten direkt „Nachbarn“ hat, über und mit denen er kommunizieren kann.

Innerhalb eines Clusters kennt jeder Knoten jeden, und jeder Knoten speichert dieselben Daten, also können Cluster von Knoten, die sich in einem anderen Cluster befinden als relativ uniforme Struktur betrachtet werden, was mit sich bringt, dass Daten nur noch auf einem Cluster abgespeichert werden und ein Routing-Algorithmus hat auch nur noch ein Cluster als Ziel.

3.1 Datenspeicherung

Die Daten werden jeweils in einem Cluster gespeichert, wodurch die Redundanz gegeben ist, da es sehr unwahrscheinlich ist, dass alle Knoten eines Clusters ausfallen (siehe Clustergröße 3.2) die Daten werden auf dem Gerät des Besitzers mithilfe eines Benutzerpassworts und dem sehr performanten „AES“-Encryption-Algorithmus[4] verschlüsselt und der Dateiname wird mithilfe des SHA-256, eines Benutzernamens, und dem Passwort einzigartig gemacht und der Dateiname wird geheim gehalten.

In welchem Cluster die Daten gespeichert werden, wird zufällig ausgesucht und diese Entscheidung wird im eigenen Cluster verschlüsselt gespeichert. Da das speichernde Cluster zufällig ausgewählt wird, wird die Datenspeicherung recht gleich auf alle Cluster verteilt, obwohl die zu Beginn beigetretenen Cluster etwas mehr Speicherplatz aufbringen müssten, da wenn wenige Cluster existieren, die Wahrscheinlichkeit pro Cluster größer ist.

Sobald ein Knoten ein zu speicherndes Datenpaket erhält, teilt er dies allen anderen Knoten des Clusters mit, die das Datenpacket ebenfalls abspeichern.

Die Daten werden auf den Knoten mithilfe einer „Hash-Map“ gespeichert, um schnelle Zugriffszeiten zu ermöglichen.

[4] Bruce Schneier, Performance Comparison of the AES Submissions: https://www.schneier.com/academic/paperfiles/paper-aes-performance.pdf (17.01.2019)

3.2 Clusterstruktur und Routing

Die Cluster untereinander verhalten sich ähnlich wie beim „Chord“-System, kennen also ihren direkten Nachfolger und Vorgänger, und mehrere Abkürzungen.

Die Abkürzungen (siehe Abb. 2) sollen einen zu langen Routingweg von $O(n)$ vermeiden und funktionieren, in dem sie $log_2(c)$ (c: Anzahl der Cluster, also $c = \frac{n}{k}$, wenn : Anzahl der Knoten und k: Anzahl der Knoten pro Cluster), Abkürzungen kennen, was zum einen sehr speichereffizient ist und, da die Cluster sortiert sind eine Routingkomplexität von maximal $O(log_2(c))$ zulässt. Insgesamt beträgt die Speicherkomplexität für einen einzelnen Knoten also $O(log_2(c) + k)$.

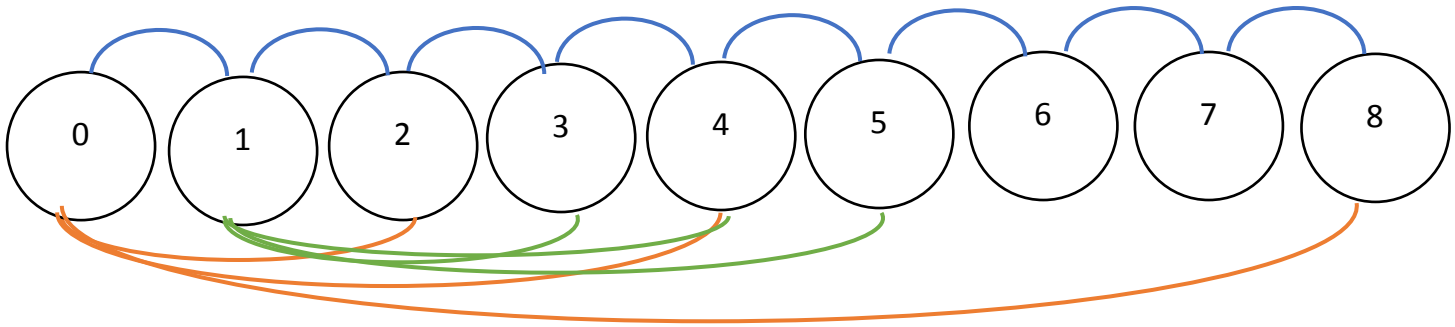

Abb. 2

Wenn im Beispiel von Abbildung 2 also Ein Datenpacket vom Cluster 0 zu 7 gesendet werden müsste, könnte wäre der Optimale Weg von 0→8→7, lediglich zwei Schritte lang. Ein Cluster kennt also immer dann ein anderes, wenn $log_2(l - a) = \mathbb{N}$ (a: eigene Clusternummer, l: Länge der Kette bis zu einem weiteren Cluster) erfüllt ist.

Die Verbindungen bestehen aus vielen Verbindungen, die Knoten innerhalb der Cluster mit denen des anderen schließen, indem jeder Knoten jeweils einen anderen aus dem verbundenen Cluster kennt (siehe Abb. 3).

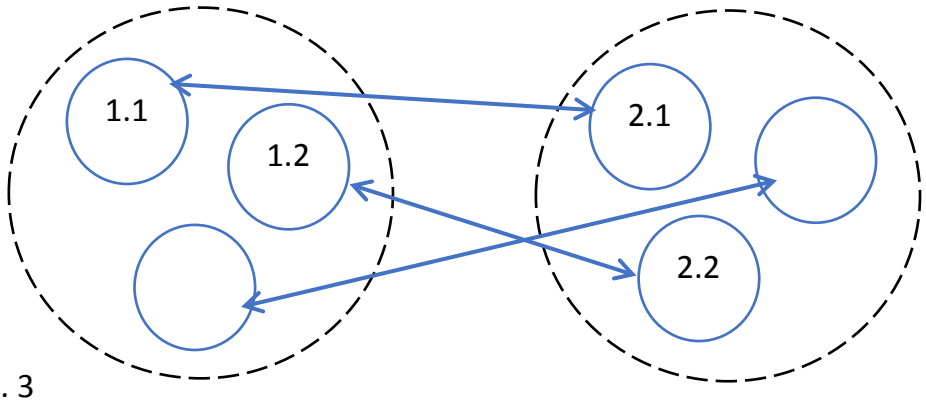

Abb. 3

Wenn ein Knoten 1.1 eine Verbindung mit dem 2. Cluster herstellen möchte, aber Knoten 2.1 momentan inaktiv ist, kann er, da sich alle Knoten innerhalb eines Clusters kennen, seine Daten über Knoten 1.2 an 2.2 senden, was trotz des Ausfalls die Routingkomplexität nur um einen Schritt erhöht.

Die verschiedenen Cluster sind hier im Gegensatz zu „DHT's" als Liste angeordnet, da bei einem Kreis (Cluster 0 stets mit dem größten Cluster verbunden) entweder das 0. Cluster sich mit jedem neuen Knoten verbinden muss, oder durch die natürlich Verteilung der Hash-Lücken zwischen den Clusternamen entstehen.

3.3 Neue Knoten hinzufügen

Um in ein neues Cluster aufgenommen zu werden, muss ein Knoten eine IP-Adresse eines bereits im Netzwerk integrierten Knotens kennen, der den neuen Knoten an das letzte Cluster weiterleitet. Im letzten Cluster wird entschieden, ob der neue Knoten in das bereits bestehende Cluster integriert wird oder, wenn das Cluster bereits k groß ist, ein neues Cluster bildet.

3.3.1 Integrieren

Der neue Knoten erhält von einem zufälligen Knoten im Cluster alle Informationen über andere Knoten, die dieser kennt und gibt die Informationen über den neuen Knoten an alle übrigen Knoten innerhalb des Clusters weiter. Der neue Knoten fordert nun von allen „Abkürzungsknoten" *einen* Knoten an, der noch keine Verbindung zu seinem neuen Cluster hat und speichert diesen ab.

3.3.2 Neues Cluster bilden

Wenn das letzte Cluster bereits voll ist, dann wird eine Verbindung zu einem der Knoten in diesem Cluster hergestellt, der anschließend seinen Wert für ändert und dies an alle seine „Abkürzungsknoten" weitergibt. Die Knoten überprüfen jeweils, ob die Bedingung $log_2(l - a) = \mathbb{N}$ erfüllt ist und fügt den neuen Knoten gegebenenfalls als Abkürzung hinzu.

3.3.3 Finden eines Knotens

Man muss um den Netzwerk beitreten zu können eine IP-Adresse, eines Knotens kennen, diese kann man erfahren, indem man einen Nutzer kennt und dessen Adresse erfragt oder über eine „Mirror-Website", ähnlich zu „Gnutella"[5], die die Adressen von zufälligen Knoten teilt und einen als initialer Knoten aussucht.

[5] Stephan Brumme, Monitoring The Gnutella Network: http://www.stephan-brumme.com/download/agents/Intelligent%20Agents.pdf (17.01.2019)

3.4 Löschen von Daten

Um Daten aus einem Cluster zu löschen, wird ein Knoten innerhalb dieses Clusters damit beauftragt, an alle Cluster-Knoten weiterzugeben ihre Version der Knoten zu löschen. Damit nicht jeder die Macht hat Daten aus einem Cluster zu löschen, besitzen die Daten einen „Besitzer-Hash“, der in dem Lösch-Datenpaket enthalten ist und der mit dem er gespeichert ist abgeglichen wird. Sind die beiden Hashes gleich, dann wird die Datei gelöscht.

3.5 Rekonfigurieren der Knotendaten

Angenommen ein Knoten ist inaktiv, während eine Datenänderung innerhalb eines Clusters stattfindet, dann verfügt dieser Knoten beispielsweise noch über eigentlich gelöschte Daten oder speichert noch keine Daten, die dem Cluster übergeben wurden.

Deshalb fragt ein Knoten bei der erneuten Aktivierung alle Knoten aus demselben Cluster nach deren Hashes der Knotendaten, wenn die Mehrheit der Knoten einen anderen Hash hat, dann werden die eigenen Knotendaten an die der Mehrheit angeglichen.

3.6 Umgang mit dynamischen IP-Adressen

In einer gewissen Frequenz f vergleicht ein Knoten seine bisherige IP-Adresse mit seiner aktuellen, wenn sich die beiden unterscheiden, dann wird die aktuelle IP-Adresse an alle bekannten Knoten geschickt, die diese dann aktualisieren.

3.7 Daten abrufen

Wenn sich der Nutzer am selben Gerät befindet, mit dem die Datei gespeichert wurde, dann wird der Speicherort aus dem Cluster entnommen (siehe 3.1) und eine Anfrage an das Speichercluster gesendet (siehe 3.2). Da diese Anfrage die eigene IP-Adresse beinhaltet, werden die Daten auf direktem Wege mit der konstanten Routingkomplexität $O(1)$ zurückgesendet.

Wenn sich der Nutzer an einem anderen Gerät befindet, dann muss zu Beginn die Clusternummer, mit der die Daten gespeichert wurden angegeben werden, um das ursprüngliche Cluster und damit die Daten zu finden.

Außerdem müssen Passwort und Nutzername angegeben werden, um die Datei nach dem Eintreffen zu entschlüsseln und den Namenshash zu generieren, mit dem die Datei gefunden werden kann.

4. Umsetzung/ Ergebnisse

4.1 Das Programm

Als Programmiersprache wird Java genutzt, da Java die meistgenutzte Programmiersprache weltweit ist[6], der Code nicht für jeden Typ von Nutzergerät neu kompiliert werden muss und es einfacher ist eine Android-App mit Java-Code zu erstellen als mit anderen Programmiersprachen. Allerdings stieß ich auf ein Problem, dass man seine eigene Remote-IPv6-Adresse nicht ohne einen von einer 3. Partei gestellten Service erfahren kann, weshalb für das Herausfinden der IP-Adresse vorerst ein „Ruby-Script“ verwendet wurde.

Beim Ausführen des Programms werden zunächst alle Speicherorte mit Dateinamen abgerufen, um die Nutzererfahrung wie bei dem vom Betriebssystem bekannten Dateiensystem zu machen.

4.2 Verwendeter Speicher

Das Java Programm, als „.jar“, inklusive dem „Ruby-Script“ ist etwas größer als 10 Kilobyte, was zum Beispiel kleiner ist als dieses „.docx“- Dokument und damit keine Belastung für Nutzer mit älteren Geräten darstellen sollte.

Neben dem Programm selbst und den von Nutzern abgespeicherten Daten, verfügt ein jeder Knoten über eine Liste von bekannten Knoten, die die Knoten-ID, IPv6-Adresse, den Port und die Clusternummer abgespeichert.

Da eine IPv6-Adresse 16 Byte groß ist und ein Integer 4 Byte[7], ist diese Liste theoretisch $= (k + log_2(c)) * (4 * 3 + 16) = (k + log_2\left(\frac{n}{k}\right)) * 28.$

So müsste ein Knoten bei einer Clustergröße von 10 Knoten und insgesamt eine 1.Millionen Nutzern ca. 475Byte $\approx \left(10 + log_2\left(\frac{1000000}{10}\right)\right) * 28)$ = 0,75Kilobyte abspeichern, was sehr gering ist. Diese Formel stimmt ungefähr, obwohl der tatsächliche Wert etwa 100 Byte größer ist, da die Daten in einer „Hash-Map“ abgespeichert werden, die auch ein wenig Speicher benötigt.

Die Größe der gespeicherten Nutzerdaten lässt sich nicht berechnen, da diese vom Nutzerverhalten abhängig ist und sich die einzig auffindbare Studie hinter einer Paywall befindet[8].

---

[6] Stackify, A Look At 5 of the Most Popular Programming Languages of 2019: https://stackify.com/popular-programming-languages-2018/ (17.01.2019)

[7] Oracle, Primitive Data Types: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html (17.01.2019)

[8] ACM, Cumulus: Filesystem backup to the cloud: https://dl.acm.org/doi/10.1145/1629080.1629084 (17.01.2019)

Code: https://github.com/f-eliks/Dezentralisierung-von-Datenspeicherung

5. Ergebnisdiskussion

Die Umsetzung erweist sich als sehr effektiv und nimmt abgesehen von den Nutzerdaten sehr wenig Speicherplatz ein.

Auch wenn bei schlechter Internet Verbindung kleine (unter 500 Kilobyte) Dateien einfach und schnell verschickt und empfangbar sind, ist die Downloadgeschwindigkeit von den Knoten abhängig, auf denen die Daten gespeichert sind. Dies ist allerdings kein grundlegendes Problem, da man das Routing (siehe 3.2) auch abhängig von den jeweiligen Netzwerkgeschwindigkeiten machen könnte, also beim Routing schnelle Verbindungen langsamen vorzieht.

Außerdem ließe sich dies mit „Sharding“, also dem Aufspalten von größeren Dateien in kleinere Datenpakete, lösen, da eine schlechte Internetverbindung bei kleinen Datenmengen kaum negativ auffällt.

5.1 Clustergrößen

Die Clustergröße k ist dafür verantwortlich, wie redundant die Daten sind und ist von der Art der Knoten abhängig. Wenn es sich um stabile Server handelt, die immer aktiv sind, genügt $3 < k < 7$.

Wenn es sich allerdings um Nutzergeräte handelt, die teils mobil sind, sollte $15 < k < 30$ zutreffen, da die Wahrscheinlichkeit von inaktiven Geräten sehr viel höher ist.

5.2 Nutzer vergisst sein Passwort

Da die Daten, sobald sie einmal verschlüsselt sind nur noch mit Passwort zu entschlüsseln sind, gehen alle Daten irreversibel verloren, wenn das Passwort vergessen wird. Außerdem darf auch der Nutzername nicht vergessen werden, da die Dateien sonst nicht auffindbar sind.

5.3 Löschen der Daten

Da der Besitzerhash für alle Knoten gleich ist, könnte ein Knoten innerhalb des Speicherclusters eine Datei auf allen Knoten eines Clusters löschen.

Um dies zu vermeiden, muss der Besitzer-Knoten individuelle Hashes erstellen, was es unmöglich machen würde, dass die Dateien von anderen als dem Besitzer gelöscht würden. Allerdings macht dies das Routing (siehe 3.7) komplizierter machen.

6. Fazit

Das Projekt ließ sich grundsätzlich gut umsetzen und alles zu Beginn Vorgenommene konnte erreicht werden. So eignet es sich um Daten in vielen Anwendungsbereichen mit der Erfüllung des Ziels: „Daten dezentralisiert, garantiert sicher, anonym und redundant zu speichern“, wie beispielsweise bei Endverbraucher-Cloud-Systemen aber auch Datenspeicherung für Firmen, die einen großen Wert auf die Zuverlässigkeit ihrer Daten legen und nicht für viel Geld mehrere (wegen der Redundanz) große Datenzentren nutzen wollen, die sich außerhalb befinden. Das System ließe sich auch als „Backend“ einer Website verwenden.

Allerdings zeigte sich auch, dass an manchen Dingen, an die zu Beginn nicht gedacht wurde, wie die Geschwindigkeiten und das „Sharding“ (siehe 5), noch gearbeitet werden muss.