Proseminar

P2P Systeme

Autonomous Replication for High Availability in Unstructured P2P Systems

Francisco Matias Cuenca-Acuna, Richard P. Martin, Thu D. Nguyen

Hristo Pentchev

20. Januar 2005

Inhaltsverzeichnis

Kapitel 1

Einführung

P2P Systeme sind so definiert, dass jedes Mitglied jeder Zeit die Systeme verlassen kann, ohne sich darum kümmern zu müssen, welche Auswirkungen dies auf die anderen Peers hat. So kommt es oft dazu, dass Informationen, die im Systeme gebraucht werden, plötzlich nicht mehr zur Verfügung stehen. Wenn dieser Vorfall im Kazaa betrachtet wird, ist der Verlust von Daten nicht sehr bedeutend. Die P2P Anwendungen gehen aber heutzutage über die "MP3-Welt" hinaus. Deswegen wird es immer notwendiger, dass die Verfügbarkeit von Daten in P2P Systeme höher wird. Sie ist selbstverständlich von der Erreichbarkeit der Peers abhängig. Da diese nicht von einem Algorithmus zu steuern ist, wird ein anderer Weg gesucht.

1.1 Naive Replikation

Die naive Idee ist, die Dateien mehrfach zu kopieren und im System zu verteilen. Wenn mehrere Peers ein und dieselbe Datei haben, dann wird mit höherer Wahrscheinlichkeit zu einem Zeitpunkt eine von denen online sein. Es wird ein Beispiel betrachtet, um die Ergebnisse der naiven Replikation zu ermitteln.

- Beispiel

  Es wird ein P2P System genommen, mit 20% durchschnittlicher Peer Erreichbarkeit. Die Dateien im System werden 6-mal kopiert. Dabei wird zusätzlicher Speicherplatz(excess storage) benötigt, dieser wird als Ratio zwischen dem gesamten Speicherplatz, der benutz wird, und der Größe aller Daten dargestellt(in unserem Bsp. haben wir 7X zusätzlichen Speicherplatz). Die Erreichbarkeit der Daten ergibt sich aus der Wahrscheinlichkeit die Daten im System zu finden. Diese ist gleich 1-die Wahrscheinlichkeit, dass die Daten nicht im System sind. Bei der naiven Replikation ist eine Datei nicht mehr im System zu finden, wenn alle Knoten, die die Datei haben, nicht online sind.
  $A = 1 - 0,8^7 = 1 - 0,2097 = 0,79$

Das Beispiel zeigt, dass, trotz der Größe der Speicher, die Ergebnisse von der naiven Replikaton nicht gut genug sind. Es wird also sehr viel Speicher benötigt, um mit naiver Replikation 99,9% Verfügbarkeit der Daten zu bekommen.

1.2 Erasure Coding

Um den Nutzen vom Speicher zu optimieren werden nicht die ganzen Dateien verteilt, sondern nur Fragmente von ihnen. Die Art der Fragmentierung, die benutzt wird, wird Erasure coding genannt. Beim Erasure coding wird ein File zuerst in m Teile fragmentiert, diese werden dann in Teile codiert, wobei $n > m$ gilt und um das File zu reproduzieren genügen m verschiedene Teile von den n. Dadurch wird weniger Information durch das System herum geschickt und das Ausfallen von einzelnen Peers hat geringere Auswirkungen auf die Reproduktion des Files.

Im Kapitel 2 wird PlanetP vorgestellt und in Kürze gezeigt, auf welche Art die Peers im Systeme miteinander kommunizieren. Darauf folgend wird in Kapitel 3 ein Replikation-Algorithmus ausführlich dargestellt. Dieser Algorithmus wird beim Ausführen von Simulationen in verschiedenen P2P Systeme eingesetzt. Die Ergebnisse von diesen Experimenten werden in Kapitel 4 präsentiert. Das Ende stellt eine Zusammenfassung der Arbeit dar.

Kapitel 2

PlanetP

Für die Art von Replikation, über die in Kapitel 3 berichtet wird, wird ein geringes Wissen benötigt. In diesem Kapitel werden die Mechanismen, die dieses Wissen verlangt vorgestellt. Sie werden nicht bis ins Detail studiert, weil sie nicht im Mittelpunkt des Themas stehen.

PlanetP ist eine Infrastruktur, die auf Gossiping basiert. Gossiping ist der zufällige Nachrichtenaustausch zwischen den Mitgliedern einer Kommune. Das Gossiping dient dazu, die Information im System aktuell und schwach konsistent zu halten und auch zum Replizieren von Datenstrukturen. Bei unserem Ansatz werden durch Gossiping auch die Fragmente transportiert.

Die Peers im PlanetP warten zwei Datenstrukturen, ein globaler und ein lokaler Index. Jeder Peer hat seinen eigenen lokalen Index. Der globale Index im System ist ein und wird auf allen Peers repliziert. Die Kopien vom globalen Index werden über Gossiping verteilt.

Der lokale Index beschreibt den Inhalt von dem Peer, an dem er angelegt ist. In dem sind die Dokumente beschrieben, die für die anderen Mitglieder freigegeben sind.

Der globale Index beinhaltet Informationen über den Zustand des Systems. Sein wichtigster Teil sind die Abbildungen $term \rightarrow peer$. Die werden dann im Index eingetragen, wenn der Peer p mindestens ein Dokument, in dem t auftritt, freigegeben hat.

Die Rolle von diesen Abbildungen und von den anderen Informationen im globalen Index in unserem Algorithmus wird im nächsten Kapitel erklärt.

Kapitel 3

Der Algorithmus

3.1 Annahmen und Terminologie

In Kapitel 1 wurde erklärt, dass die Erreichbarkeit von Daten in P2P Systemen ein aktuelles Problem ist. Weiter wurde gezeigt, dass eine Möglichkeit, dieses Problem zu lösen, die Replikation von Daten ist und dass gute Ergebnisse mit der Replikation von Fragmenten erzielt werden. In Kapitel 2 wurde beschrieben, wie die Fragmente durch Gossiping verteilt werden. Nun wird erklärt, welche Fragmente verteilt werden. Im Folgenden wird ein Replikations-Algorithmus vorgestellt, der die Verfügbarkeit von Daten erhöht. Der Algorithmus ist so definiert, dass er ein geringes Wissen über das System braucht, um zu funktionieren. Das Ziel ist, zu zeigen, dass so ein dezentralisierter Algorithmus in nicht strukturierten P2P Systemen brauchbare Ergebnisse erzielt. Der Algorithmus regelt das Verhalten von Peers, die sich im PlanetP System befinden. PlanetP liefert dem Algorithmus die Informationen über den Zustand des Systems, die er braucht. Um den Algorithmus zu beschreiben, müssen ein paar Definitionen von Begriffen und Annahmen eingeführt werden.

Die erste Regel ist, dass Peers nur dann eine ganze Datei speichern, wenn sie sie für offline Operationen brauchen. Alle ganzen Files, die ein Peer hat, befinden sich in seinem Hoard Set.

Neben dem Hoard Set bietet jeder Peer auch Speicherplatz für Fragmente(Replication Store) an. Die Größe der Replication Stores ist begrenzt. Jeder Peer kann also nur eine bestimmte Menge von Fragmenten speichern. Die Benutzung der Replication Stores zu regeln ist eines der wichtigsten Ziele des Algorithmus.

Alle Files haben eine eindeutige ID, die dazu dient, dass man aktuelle Informationen über die Files im globalen Index sammeln kann.

Wenn über die Verfügbarkeit von einem Fragment gesprochen wird, wird eigentlich die Verfügbarkeit von dem File gemeint, zu dem das Fragment gehört.

Im Weiteren wird zwischen Peers, die Fragmente rausschicken und Peers, die sie empfangen, unterschieden. Peers, die Fragmente rausschicken, werden Replicators genannt. Peers, die Fragmente bekommen, werden Targets genannt.

3.2 Der Algorithmus

Der Algorithmus besteht aus vier Punkten:

- Jeder Peer trägt die IDs von den Files und Fragmenten, die er gespeichert hat, in den globalen Index ein.
- Von jedem Peer wird periodisch die Verfügbarkeit der gespeicherten Daten gerechnet.
- Jeder Peer generiert periodisch ein Fragment von beliebigem File in seinem Hoard Set und schickt es zu einem per Zufall ausgesuchten Target.
- Der Target Peer wird entweder das Fragment speichern oder es ablehnen.

Die drei wichtigsten Punkte des Algorithmus sind das Berechnen der Verfügbarkeit, die Replicators- und die Targets-Tätigkeiten. Auf diese wird nun nher eingegangen.

3.3 Berechnung der Verfügbarkeit

Wie bereits in Kapitel 1 erwähnt wurde, ist die Erreichbarkeit von einem File bei der Naive Replikation gleich (1 - die Wahrscheinlichkeit, dass alle Knoten, die dieses File besitzen, offline sind). In unserem Fall kann der File auch von Fragmenten reproduziert werden. Dazu, dass alle Peers, die das File im Hoard Set gespeichert haben, offline sind, müssen auch genügend viele Knoten, die Fragmente des Files haben, auch offline sein. Mit "genügend viele" ist hier gemeint, dass weniger als m Fragmente im System zu finden sind(m Fragmente reichen aus, um das File zu reproduzieren). Wenn also von einem File n Fragmente schon verteilt sind, um das File nicht reproduzieren zu können, müssen mindestens $(n - m + 1)$ Fragmente nicht verfügbar sein. Um die Erreichbarkeit von Daten zu berechnen, wird die Erreichbarkeit der Peers benötigt, deswegen trägt jeder Knoten seine Online- und Offline-Zeiten in den globalen Index ein. Neben diese Informationen teilt jeder Peer den anderen mit, welche Fragmente und Files sich bei ihm befinden. Das wird gemacht, indem die Abbildungen $File\_Hash(f) \rightarrow m$ und $Frag\_Hash(f) \rightarrow m$ in den/aus dem globalen Index ein-/ausgetragen werden. $File\_Hash(f) \rightarrow m$ sagt aus, dass File f im Knoten m gespeichert ist. Analog bedeutet $Frag\_Hash(f) \rightarrow m$, dass Fragmente vom File f beim Knoten m zu finden sind. Für die Berechnungen wird angenommen, dass jeder Peer zu jedem Zeitpunkt entweder das ganze File oder nur ein einziges Fragment vom File gespeichert hat. Mit diesen Informationen können die Knoten, die das File f haben in eine Menge $H(f)$ zusammengefügt werden und die, die über ein Fragment des Files f verfügen, in $F(f)$.

Wegen der großen Anzahl von Fragmenten wird bei der Berechnung eine Vereinfachung vorgenommen. Es wird P_{avg} eingeführt, welches die durchschnittliche Wahrscheinlichkeit beschreibt, dass Knoten von $F(f)$ offline sind.

$$H(f) = \{m \mid (File\_Hash(f) \rightarrow m) \in GlobalIndex\}$$

$$F(f) = \{m \mid (Frag\_Hash(f) \rightarrow m) \in GlobalIndex\}$$

$$P_i = \frac{avg.offlineTime}{avg.onlineTime + avg.offlineTime}$$

$$P_{avg} = \frac{1}{n} \sum_{i \in F(f)} P_i$$

$$A(f) = 1 - \prod_{i \in H(f)} P_i \sum_{j=n-m+1}^{n} \binom{n}{j} P_{avg}^{j} (1 - P_{avg})^{n-j}$$

Abbildung 3.1:

3.4 Generieren und Senden von Fragmenten

Jeder Knoten verfügt also über das Wissen, welche Erreichbarkeit die Files in seinem Hoard Set haben. Jeder Peer wird versuchen die Verfügbarkeit, von den Dateien, die nicht die gewünschte haben, zu erhöhen. Er wird sich irgendeines von diesen Files aussuchen und wird genau ein beliebiges Fragment produzieren. Dieses wird dann zu einem per Zufall gewählten Target verschickt. Dabei wird eine Art Erasure Coding benutzt, bei der n viel größer m ist ($n >> m$). Diese Tatsache und die zufällige Auswahl des Files und des Fragments garantieren uns, dass mit einer geringen Wahrscheinlichkeit zwei gleiche Fragmente produziert werden. Für unsere Wahl von Codierung spricht auch das Steigen vom Nutzen von Fragments zusammen mit dem vergrößern von n stattfindet. So werden auch in größeren P2P Systemen stabile Werte erreicht, selbst, wenn viele Peers plötzlich das System verlassen.

3.5 Empfangen von Fragmenten

Die Last der Entscheidungen liegt bei den Targets. Wenn ein Fragment bei einem Target mit freiem Platz im Replication Store ankommt, wird dieses ge-

speichert. Wenn der Replication Store voll ist, muss entschieden werden, ob das neue Fragment verworfen wird oder ob mit ihm eins von den alten ersetzt wird. Die Entscheidung soll so getroffen werden, dass maximale Effizienz gewonnen wird. Es ist nicht sinnvoll Fragmente zu verlieren, die kleine Verfügbarkeit haben oder solche zu speichern, die relativ hohe Verfügbarkeit haben. Deswegen wird eine Eintrittsgrenze berechnet und nur Fragmente, die Verfügbarkeit unter diese Grenze haben werden gespeichert. Diese Grenze ergibt sich aus den Verfügbarkeiten der Fragmente im Replication Store und beträgt 10 Prozent über dem Durchschnittswert. Wenn die Verfügbarkeit des neuen Fragments unter der Grenze liegt, muss ein Fragment aus dem Store rausgeworfen werden. Eigentlich ist es intuitiv klar, dass das Fragment mit der höchsten Verfügbarkeit raus muss. Wenn aber so eine deterministische Vorgehensweise benutzt würde, könnte es passieren, dass in kurze Zeit Fragmente demselben File von vielen Peers verworfen werden. Dies würde zu Schwankungen bei Verfügbarkeitswerten führen. Um das zu vermeiden wird gewichtete Lotterie durchgeführt. Den Fragmenten werden Lose zugeteilt. Diese werden zuerst in zwei Mengen geteilt. Die größere der beiden Mengen hat 80% von den Losen, die kleinere 20%. Diese 20% werden gleichmäßig zwischen den Fragmenten verteilt. Die 80% werden nur zwischen den Fragmenten verteilt, deren Verfügbarkeit größer als die Grenze(Durchschnittswert + 10%) ist. Der Gewinner der Lotterie wird verworfen und das neue Fragment kann gespeichert werden. Im Anhang ist ein Beispiel zu finden, das dieses Glückspiel veranschaulichen soll.

3.6 Optimierungen

Wenn also der Replicator das falsche Target(Target mit voll belegtem Speicherplatz) ausgewählt hat wird das Fragment, das er geschickt hat, wahrscheinlich abgelehnt. Ein Ansatz den Algorithmus zu optimieren wäre, dass Replicators sich ein Target mit freiem Platz im Replication Store aussuchen. So eine Optimierung könnte aber dazu führen, dass viele Fragmente zum selben Target, wenn das ein von den letzten Peers mit freiem Platz im Replication Store ist, gesendet werden und es damit überfordern.

Weitere Optimierung wäre das Hinzufügen von Lotterie beim Replicator, die analog zu der Lotterie beim Target funktioniert. Diese Verbesserung wird zur Erhöhung der Minimum Verfügbarkeit führen. Sie wird aber rechnerisch das System zusätzlich belasten. Es wäre interessant zu beobachten, ob sie die Wahrscheinlichkeit von Wiederholungen von Fragmente deutlich vergrößert.

3.7 "Böse"Peers

Der Algorithmus funktioniert auch, wenn sich im System Peers befinden, die ihn nicht einhalten wollen. Die Autoren des Papers haben 3 verschiedene Szenarios betrachtet um dies zu zeigen.

Als erstes werden Peers betrachtet, die Fragmente im Replication Store

beschädigen. Dieses würde zu Verlust von Speicher und auch zu niedrigerer, als die berechnete, Verfügbarkeit führen. Für die Vollständigkeit des Files hätte es aber keine Auswirkung, weil im Erasure coding solche Fehler entdeckt werden.

Das Zweite Beispiel sind Peers, die versuchen würden Fragmente von Files zu senden, die bereits eine hohe Erreichbarkeit haben. Dies würde außer der Verschwendung von Bandbreite keine Auswirkungen haben, weil alle Entscheidungen von dem Target getroffen werden.

Peers könnten auch im globalen Index falsche Informationen über sich annoncieren. Der schlimmste Fall wird unser drittes Beispiel. Er würde dann auftreten, wenn ein Peer X von sich behaupten würde, dass er immer online ist, und dass er eine große Menge von Files im Hoard Set bewahrt. Das wird alle anderen Peers davon abbringen diese Files zu replizieren. Welches dem Peer X die Möglichkeit schaffen würde, Replication Store von anderen Peers für sich zu gewinnen. Dieses Vorgehen ist aber für X nicht im eigenen Interesse, weil viele Peers ihm Anfragen senden würden, ausgehend von seinen falschen Informationen. Welches ihn Bandbreite kosten würde.

Kapitel 4

Simulationen

Nachdem die Idee unseres Ansatzes vorgestellt wurde, ist es Zeit zu sehen ob sie auch wie gewünscht funktioniert. Um die Effektivität des Algorithmus zu ermitteln wurden Simulationen durchgeführt. Für die Simulationen wurde ein Simulator gebaut, der unter etwas idealisierten Bedingungen funktioniert. Im Folgenden werden die Annahmen für die Experimente beschrieben und die Ergebnisse analysiert.

- Alle Peers werden gleichzeitig Fragmente rausschicken.
- Es wird angenommen, dass alle Fragmente auch gleichzeitig ankommen. Die Zeit für Datenübertragung wird nicht berücksichtigt.
- Außerdem wird das Reproduzieren vom Globalen Index durch Gossiping ignoriert.
- Das Auswerten von Datenverfügbarkeit wird alle 10 Minuten ausgeführt. Welches die Peers mit nicht aktuellen Daten arbeiten lässt.
- Für Erasure coding wird ein konstantes m (m=10) benutzt. Damit wird die Fähigkeit des Erasure coding, das m optimal zu bestimmen, außer Gefecht gesetzt.

Alle Vereinfachungen führen dazu, dass die Arbeit des Simulators nicht so genau abläuft, wie der Algorithmus es beschreibt. Die Ergebnisse werden aber dadurch realistischer, denn allein das Realisieren der Lotterie ein hohen rechnerischen Aufwand beinhaltet.

Die Experimente wurden mit drei verschiedenen P2P Systemen durchgeführt. Die Systeme unterscheiden sich unter anderem um die folgenden Merkmalen:

- Die Anzahl der Mitglieder
- Die Anzahl der Files im System
- Die Größe der Files

- Die Online Zeiten der Peers

Die Parameter sind in der Abbildung 4.1 beschrieben.

	File Sharing	Corporate	Workgroup
No. Members	1 000	1 000	100
No. Files	25 000	50 000	10 000
Node availability	24,9%	80,7%	33,3%

Abbildung 4.1:

Die Resultate der Simulationen führen zu manchem Rückschlüssen über die Beziehung zwischen den Systemcharakteristiken und die Effektivität des Algorithmus. Anschließend wird der präsentierte Algorithmus (weiter REP genannt) mit anderen verglichen, um noch mal seine Funktionalität zu studieren.

4.1 Durchschnittliche Peer-Verfügbarkeit

Sowohl bei den Berechnungen (Abbildung 4.2), die uns eine Vorstellung über die zu erwatende Werte liefern, als auch bei den Simulationen (Abbildung 4.3) ist die Rolle von einer hohen durchschnittlichen Peer-Verfügbarkeit bemerkbar. CO(durchschnittliche Peer-Verfügbarkeit 80%) brauch wesentlich weniger zusätzlichen Speicherplatz als FS(25%) oder WG(33%).

	excess storage	availability
CO	1,75X	3 nines
WG	6X	3 nines
FS	8X	3 nines

Abbildung 4.2:

4.2 Server-like Peers

Interessant, zu sehen, ist, dass sich die Werte bei FS und WG von den Berechnungen und von den Simulationen vertauscht haben. Dies liegt daran, dass bei den Experimenten der Vorteil von den so genannten Server-like Peers, solche Knoten die eine sehr hohe Verfügbarkeit haben, ausgenutzt wird. Welches bei

	excess storage	availability
CO	1X	3 nines
WG	9X(uniform)	3 nines
FS	6X	3 nines

Abbildung 4.3:

den Berechnungen nicht der Fall ist. Daten die sich auf diesen Peers befinden, gewinnen dadurch eine hohe Verfügbarkeit und müssen deswegen kaum repliziert werden. Das entlastet das System und spart Speicherplatz.

4.3 Zentrales Wissen gegen autonome Entscheidungen

Beim direkten Vergleich von REP mit einem Algorithmus(OMNI), der über perfektes zentrales Wissen verfügt, wird klar, dass OMNI bessere Werte erzielt. Da OMNI nicht in der realen Welt zu implementieren ist, und weil der Unterschied nicht so groß ist, wird der Einsatz von einem autonomen Algorithmus gerechtfertigt.

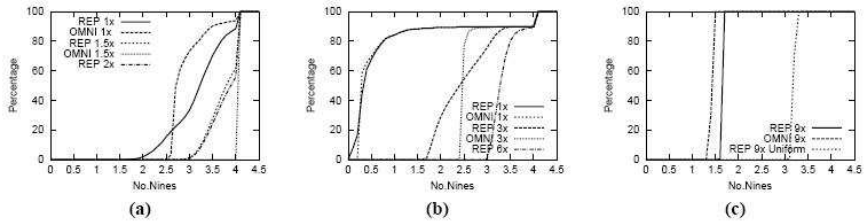

Figure 3. *CDFs of file availability for (a) the CO community with 1X, 1.5X and 2X, (b) the FS community with 1X, 3X, and 6X, and (c) the WG community with 9X excess storage capacities. REP refers to our replication algorithm as described in Section 3. OMNI is as described in Section 4.3.*

Abbildung 4.4:

4.4 Ersetzen basierend auf der Verfügbarkeit

REP wurde mit noch einem Algorithmus(BASE), bei dem die Fragmente im Replication Store nach der FIFO Regel ausgetauscht werden, verglichen. Dabei ist fällt auf(Abbildung 4.5), dass bei BASE 16% der Daten eine Verfügbarkeit kleiner 0,9 haben, während solche Dateien bei REP nur 1% ausmachen. Die Fairness von REP bringt also bessere Ergebnisse.

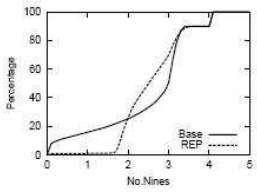

Abbildung 4.5:

Kapitel 5

Zusammenfassung

Da die Rolle von P2P Systeme immer größer wird, werden diese auch gründlich erforscht. Versucht wird es die Vorteile der Autonomie, die Peers haben, auf reale Anwendungen zu übertragen. Dadurch soll eine Alternative zu server-basierten Internet Diensten verschafft werden. Dabei treten verschiedene Probleme auf. Eins davon ist die Verfügbarkeit von Daten.

In dieser Ausarbeitung wurde eine Möglichkeit, die Verfügbarkeit von Daten in P2P Netze zu erhöhen, vorgeschlagen. Unter anderem wurden die Begriffe PlanetP, Erasure coding und excess storage eingeführt und erläutert.

Das Hauptziel war zu zeigen, welches auch gelungen ist, dass in verschiedene Netze ohne zentrales Wissen und ohne Speicherplatz zu verschwenden die Verfügbarkeit von Daten erhöht werden kann.

Bei den Experimenten wurde der positive Einfluss von einer hohen durchschnittlichen Peer-Verfügbarkeit, und von den so genannten server-like Peers, auf den Großen des gebrauchten Speicherplatzes bestätigt.

Der Algorithmus und noch mehr der Simulator funktionieren unter idealisierten Bedingungen. Ein Beispiel ist die Annahme, dass sich Daten nicht verändern, es ist kein Vergleich von Versionen oder Update nötig. Diese idealisierte Umgebung auf die reale Welt abzubilden wird der nächste Schritt in der Entwicklung der P2P Systeme.

Literaturverzeichnis

[1] F. M. Cuenca-Acuna C. Peery R. P. Martin and T. D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. 2003.

[2] F. M. Cuenca-Acuna R. P. Martin and T. D. Nguyen. Planetp: Using gossiping and random replication to support reliable peer-to-peer content search and retrieval. 2002.

[3] Francisco Matias Cuenca-Acuna Richard P. Martin Thu D. Nguyen. *Autonomous Replication for High Availablity in Unstructured P2P Systems.* 2003.