# Anfragenbasierte Datenreplikation in Peer-to-Peer-Netzen

Sven Herschel
Humboldt-Universität zu Berlin, Institut für Informatik
LFE Datenbanken und Informationssysteme
Unter den Linden 6
10099 Berlin
herschel@informatik.hu-berlin.de

**Zusammenfassung**

Peer-to-Peer-Systeme rücken derzeit verstärkt in das Zentrum der wissenschaftlichen Forschung. Das große Ziel, dezentralisierte Knoten, die ihre Ressourcen der Gemeinschaft zur Verfügung stellen, scheint aus vielerlei Gründen verlockend: Skalierbarkeit, Ausfallsicherheit, Anonymität und Unzensierbarkeit sind nur einige von ihnen. [GHIaS01]

Dennoch sieht sich insbesondere die Datenbank-Community mit ihren traditionell hohen Anforderungen an Verfügbarkeit und Datenkonsistenz in diesem Bereich großen Herausforderungen gegenüber.

Ziel dieses Beitrags ist die Vorstellung eines Modells, nach welchem Daten anfragenbasiert innerhalb eines Peer-to-Peer-Netzes repliziert werden können, wodurch die Verfügbarkeit der Daten im gesamten Netz steigt. Gleichzeitig wird durch Einführung des Konzeptes der "Referenzdatenquelle" für jedes Objekt die Datenkonsistenz gesichert.

Mit Hilfe dieses Modells können die Aspekte Lastverteilung und Datenredundanz mit der geforderten hohen Datenkonsistenz in Einklang gebracht werden.

## 1 Motivation & Einführung

Peer-to-Peer - Technologie ist bisher vor allem aus dem Bereich des Filesharing bekannt. Systeme wie Napster [Na], Gnutella [Gn00] und nicht zuletzt die hierarchischen Systeme des FastTrack - Protokolls [Fa] haben eine hohe Bekanntheit und Verbreitung innerhalb der Internet-Gemeinde erlangt.

Jedoch stehen im Filesharing-Kontext Fragen wie Datenkonsistenz und -verfügbarkeit nicht im Zentrum der Betrachtungen. Auch aktuelle Forschung, aus dem Datenbank-Bereich motiviert, plädiert für Lockerung der strengen ACID-Anforderungen, um die Vorteile der massiven Datenverteilung in P2P-Netzen nutzen zu können [HHB+03]

In diesem Beitrag wird eine Architektur vorgestellt, die die Vorteile von Peer-to-Peer-Netzen (massive Datenverteilung, Redundanz, Lastverteilung) bei Bedarf mit den oben erwähnten strengeren Anforderungen der Datenbank-Community kombiniert.

Dazu wird in Abschnitt 2 ein kurzer Überblick über die vorhandenen Ansätze gegeben, bevor in Abschnitt 3 das Konzept der anfragenbasierten Replikation vorgestellt wird. Vor der Zusammenfassung (Abschnitt 5) beleuchtet der Ausblick in Abschnitt 4 die offenen Fragen dieses Ansatzes.

**Bemerkung zur Notation:** Ein Peer-to-Peer-Netz ist aus gleichberechtigten Knoten aufgebaut, welche sowohl Client- als auch Serveraufgaben übernehmen können. Wenn in diesem Beitrag an einigen Stellen von Client und Server die Rede ist, dann ist damit der Knoten in seiner Rolle als Client bzw. Server gemeint.

# 2 Peer-to-Peer–Systeme & Datenbanken

Die Grundlage für die weite Verbreitung von Peer-to-Peer-Systemen (P2P-Systeme) legten die so genannten Tauschbörsen, die sich primär mit der Suche und Verteilung von Dateien anhand vorgegebener Anfragen beschäftigen. Bei diesen Anfragen ist das Schema vorgegeben und bezieht typischerweise nur leicht ermittelbare Datenfelder mit ein, wie zum Beispiel bei Musiktauschbörsen Dateiname, Künstler, Album [Gn00].

An den Tauschbörsen lässt sich leicht die Evolution der P2P-Systeme nachvollziehen. Während Napster noch einen zentralen Index für die Anfragen verwendete [Na] und nur der eigentliche Datenaustausch zwischen den Peers erfolgte, gingen die Nachfolger, Gnutella und FastTrack, zu einem reinen P2P-System über. Dies bedeutet, dass auch die eigentlichen Anfragen von erreichbaren Peers beantwortet werden und es somit keinen "Single Point of Failure" mehr gibt.

Dank des verwendeten Anfragealgorithmus können jedoch Anfragen nun nicht mehr vollständig beantwortet werden, da mit jeder Anfrage (nur) ein Teil des P2P-Netzes "geflutet" wird und somit Datenobjekte, die sich nicht im gefluteten Teil des Netzes befinden, nicht gefunden werden können.

Eine Lösung gegen das Flooding des Netzes und für das zuverlässige Auffinden von Objekten bieten seit kurzem so genannte "Distributed Hash Tables" (DHTs). Dabei werden alle an der DHT beteiligten Nodes zu einem virtuellen Overlay-Netzwerk zusammengeschlossen, in welchem ein sehr effizientes Routing von Nachrichten möglich ist. Trotz substanzieller Unterschiede in der Implementierung zwischen den einzelnen DHT-Systeme (siehe unten) unterstützen sie alle, direkt oder indirekt, die grundlegenden Funktionen einer Hashtable, nämlich `lookup(key)` und `put(key, value)`. DHTs wandeln also mittels einer Hashfunktion den Schlüssel (key) des gesuchten Objektes in die Adresse des Knotens um, auf dem das Objekt gespeichert ist. Somit werden diese Objekte immer gefunden, wenn ihr Name bekannt ist und der zugehörige Knoten innerhalb des Overlay-Netzwerkes erreichbar ist.

Die bekanntesten Vertreter der DHT-Algorithmen sind CAN [RFH+01], Chord [SMLN+03], Pastry [RD01] und Tapestry [ZKJ01]. Details zu den einzelnen DHTs finde sich in den jeweiligen Artikeln.

Erste Ansätze aus der Datenbank-Community, welche versuchen, die Vorteile von DHTs für der Erforschung massiv verteilter Datenbanken zu nutzen, finden sich z.B. in [HHB+03].

Alle dem Autor bekannten Ansätze stellen jedoch Fragen der Datenkonsistenz zugunsten der massiven Verteilbarkeit der gespeicherten Daten zurück.

# 3 Anfragenbasierte Datenreplikation

## 3.1 Modell

In diesem Abschnitt wird ein Modell vorgestellt, welches es ermöglicht, Dateien in P2P-Netzen anfragenbasiert zu verteilen und durch Einführung einer "Referenzdatenquelle" gleichzeitig Lastverteilung und Konsistenz zu sichern.

Dazu werden folgende Annahmen getroffen:

- Die Daten werden in einem DHT-basierten Overlay-Netzwerk abgelegt, welches das Routing von Nachrichten übernimmt.
- Die kleinste, im Rahmen des Overlay-Netzwerkes adressierbare Einheit sei eine Datei. Es ist leicht vorstellbar, diesen Ansatz auf Tabellen im Sinne relationaler Datenbanken zu übertragen.
- Jede Datei liegt auf einer so genannten Referenzdatenquelle vor.

- Eine **Referenzdatenquelle** ist derjenige Knoten, zu dem die zugrunde liegende DHT eine Anfrage nach dem Dateinamen routet. Anfragen nach der gesuchten Datei werden also immer zuerst zu der Referenzdatenquelle geroutet.

## 3.2 Anfragebearbeitung

Es sei ein P2P-Netzwerk angenommen, welches aus den Knoten A, B, C und D besteht. Auf A liegt eine Datei `Datei01` vor, für welche A Referenzdatenquelle (s.o.) sei.

Kommt nun von Knoten B eine Anfrage nach dieser Datei, wird sie von A ausgeliefert, wobei A sich B als **Replikationsknoten** in seiner Referenzliste merkt. Wenn nun C eine Anfrage nach `Datei01` an A richtet, kann A netzwerkeffizient auf B verweisen. C muss sich nun die Datei von B holen.

Bei der nächsten Anfrage, einer Anfrage von D, könnte A entweder auf B oder auf C verweisen und muss die Datei in keinem Fall selbst ausliefern.

Auf diese Art und Weise wächst die Referenzliste kontinuierlich. Die Art und Weise, wie A bei Anfragen den geeigneten Replikationsknoten auswählt, wird hier nicht weiter betrachtet. Denkbar sind jedoch Algorithmen, die die Netzwerkkapazitäten und die Auslastung der Replikationsknoten (hier B und C) in Betracht ziehen.

Bild 1 veranschaulicht die Anfragebearbeitung. Offensichtlich wandelt sich A nach diesem Modell sukzessive vom Contentprovider zum Index und liefert nach einige Zeit nur noch Redirects statt der eigentlichen Datei aus, sofern sich die Datei nicht verändert hat (siehe dazu 3.3).

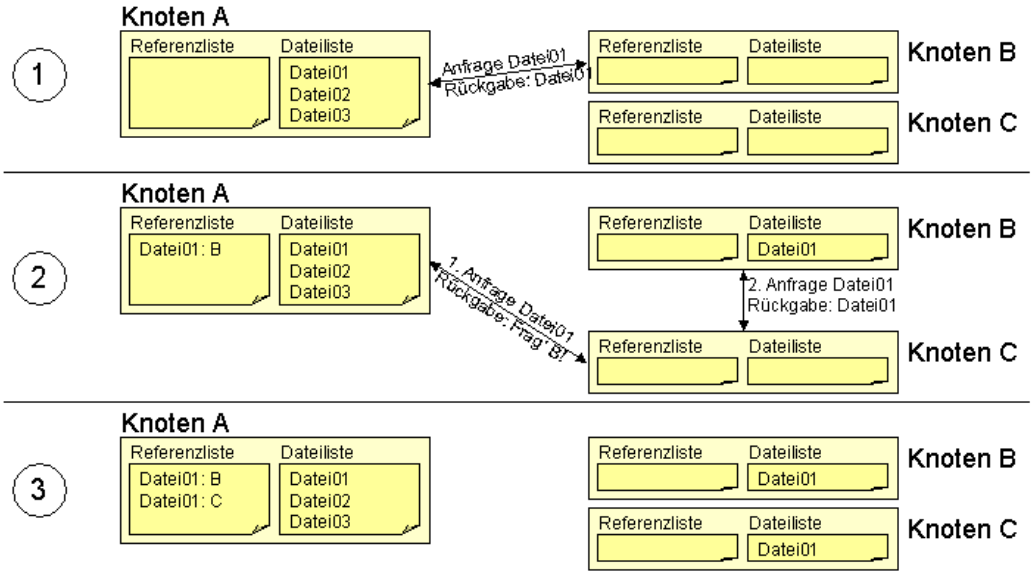


Abbildung 1: Ablauf von Anfragen mit anfragenbasierter Datenreplikation

## 3.3 Updates

Updates können nur auf der Referenzdatenquelle (in diesem Beispiel A) durchgeführt werden. Nach jedem Update wird die Referenzliste zu der aktualisierten Datei geleert. Dies führt dazu, dass der nächste anfragende Client die aktualisierte Datei direkt von A ausgeliefert bekommt und keinen Redirect auf einen anderen Knoten erhält (siehe Schritt 1 im Bild).

Auf diese Art und Weise erhält man bei Anfragen nach der Datei immer die aktuelle Version der Datei. Sollte "Update" bedeuten, dass die Datei gelöscht wurde, wird sie danach nicht mehr von der Referenzdatenquelle ausgeliefert.

# 4 Ausblick

## 4.1 Höhere Dynamik des Peer-to-Peer-Netzes

Bei höherer Dynamik des P2P-Netzes, wenn also Knoten das Netz in kurzen Abständen verlassen und ihm beitreten, müssen die Algorithmen noch einmal genauer betrachtet werden. Die folgenden Fragen, über die bisher in diesem Beitrag elegant hinweg gegangen wurde, stellen sich unmittelbar, wenn man eine höhere Dynamik des Netzes unterstellt:

**Referenzdatenquelle nicht erreichbar:** In dynamischen P2P-Umgebungen ist es sehr leicht möglich, dass die Referenzdatenquelle nicht mehr erreichbar ist. Um mit diesem Fall umgehen zu können, muss die Referenzdatenquelle ihre Daten zusätzlich dergestalt redundant ablegen, dass die DHT Anfragen bei Ausfall der Quelle zu dem redundanten Knoten leitet. Möglichkeiten hierfür bietet z.B. die DHT Pastry [RD01], wo jedem Knoten seine numerisch nächsten Nachbarn bekannt sind. Eine Duplizierung sowohl der Daten als auch der Referenzliste auf den nächsten Nachbarn führt dann automatisch dazu, dass bei Ausfall eines Knotens die Anfragen automatisch zum Nachbarn geroutet werden. In diesem Szenario sieht man sich jedoch der Herausforderung gegenüber, zwei Datenquellen gleichzeitig konsistent halten zu müssen.

**Redirect-Datenquelle nicht erreichbar:** Der derzeitige Ansatz geht davon aus, dass ein Redirect der Referenzdatenquelle auf einen erreichbaren Node zeigt. Sollte dieser jedoch das Netz bereits verlassen haben, hilft derzeit nur eine erneute Anfrage an der Referenzdatenquelle. Diese sollte den ausgefallenen Knoten dann aus ihrer Referenzliste löschen.

## 4.2 Caching und Datenaktualität

Nachdem sich im Beispiel Knoten C die Datei von Knoten B beschafft hat (Schritt 3 in der Abbildung), kann sich C merken, dass die Datei bei B vorliegt und sie im erneuten Bedarfsfalle direkt von B holen, ohne die Referenzdatenquelle A zu kontaktieren. Dies führt jedoch unter Umständen dazu, dass bei B veraltete Daten vorliegen.

Es ist nun denkbar, jeder Datei eine vergangenheitsbasierte Kennzahl der Datenaktualität hinzuzufügen, um die Wahrscheinlichkeit zu beschreiben, mit der diese Datei noch aktuell ist. Um diese Kennzahl zu berechnen, wird die bisherige Update-Häufigkeit herangezogen. Ein Client, hier C, kann dann selbst entscheiden, ob ihm die Wahrscheinlichkeit genügt oder ob er die Referenzdatenquelle kontaktieren muss. Der Kontakt der Referenzdatenquelle kann wiederum mit Kosten verbunden sein, so dass hier eine echte Tradeoff-Betrachtung durchgeführt werden muss. Meine weitere Arbeit wird sich mit dieser Frage noch detaillierter auseinandersetzen.

## 4.3 Designentscheidungen

Im Zuge einer Implementierung stellen sich einige Designentscheidungen, die sorgfältig abgewogen werden sollten:

**Welche Granularität ist für die zu speichernde Datenstruktur zu wünschen?** In diesem Beitrag wird die Datei als kleinste im Rahmen der DHT adressierbare Datenstruktur angenommen. Dies entspricht in vielen DBMSen einer Tabelle im relationalen Sinne. Es sind jedoch sowohl feinere Strukturen (Records, Teiltabellen) oder auch gröbere Strukturen (Tabellenschemata, Datenbanken) als kleinste adressierbare Einheit denkbar.

**Welche Rolle spielt geographische Lokalität?** Geht man von langen Laufzeiten der Netzwerkübertragungen aus, kann es erforderlich werden, Anfragen an die Referenzdatenquelle zu geographisch dem Client möglichst naheliegenden Servern weiterzuleiten.

### 4.4 Semantisches Routing

**Semantisches Routing** innerhalb des P2P-Netzes würde die Flexibilität bei der Anfrage der Referenzdatenquelle erhöhen. Derzeit wird davon ausgegangen, dass der Name der Datei (der `key`) bekannt ist und dass dieser eindeutig innerhalb des Netzes ist. Diese Annahmen sind gleichermaßen unrealistisch wie unpraktisch. Meine weitere Arbeit wird sich auch mit diesen Fragen auseinandersetzen. Ansätze dazu finden sich z.B. in [TXD03].

## 5 Zusammenfassung

Im Rahmen dieses Beitrags wurde ein Modell vorgestellt, das es ermöglicht, Dateien **anfragenbasiert** in P2P-Netzen zu verteilen. Dabei wandelt sich der Knoten sukzessive vom Content-Server zum Index-Server für seinen Inhalt, um die eigenen Netzwerkressourcen zu schonen.

Das Konzept der **Referenzdatenquelle** ermöglicht es zusätzlich, trotz potenziell sehr hohen Verteilungsgrades, immer auf die jeweils aktuelle Version der Datei zuzugreifen und somit den hohen Anforderungen der Datenbank-Community an die Datenkonsistenz gerecht zu werden.

## Literatur


[Fa] FastTrack: *FastTrack.* http://www.fasttrack.nu.

[GHIaS01] Gribble, S., Halevy, A., Ives, Z., und andDan Suciu, M. R.: What can databases do for peer-to-peer? In: *WebDB Workshop on Databases and the Web.* 2001.

[Gn00] Gnutella: *Gnutella Protocol Specification.* 2000. http://dss.clip2.com/GnutellaProtocol04.pdf.

[HHB+03] Huebsch, R., Hellerstein, J. M., Boon, N. L., Loo, T., Shenker, S., und Stoica, I. Querying the internet with pier. September 2003.

[Na] Napster: *Napster Homepage.* Napster, Inc. http://www.napster.com.

[RD01] Rowstron, A. und Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware).* S. 329–350. November 2001.

[RFH+01] Ratnasamy, S., Francis, P., Handley, M., Karp, R., und Schenker, S.: A scalable content-addressable network. In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications.* S. 161–172. ACM Press. 2001.

[SMLN+03] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., und Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.* 11(1):17–32. 2003.

[TXD03] Tang, C., Xu, Z., und Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications.* S. 175–186. ACM Press. 2003.

[ZKJ01] Zhao, B. Y., Kubiatowicz, J. D., und Joseph, A. D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141. UC Berkeley. April 2001.