# Replikation in Peer-to-Peer Systemen

Daniel Klan
daniel.klan@tu-ilmenau.de
Fakultät für Informatik und Automatisierung, TU Ilmenau
Postfach 100565, D-98684 Ilmenau

**Zusammenfassung**

Angesichts der zunehmenden Größe heutiger verteilter Systeme erweist sich das P2P-Paradigma als eine vielversprechende Alternative zu traditionellen Client/Server-Architekturen. Zur Verbesserung von Performance, Verfügbarkeit und Zuverlässigkeit werden die Daten in solchen Systemen auf eine Vielzahl von Peers repliziert. Das mit redundanter Datenhaltung einhergehende Problem der Konsistenzsicherung gestaltet sich in diesem Zusammenhang als Herausforderung. So sind aufgrund der Dimension von P2P-Systemen, dem Fehlen einer zentralen Instanz und ihrem dynamischen Verhalten, herkömmliche aus dem Bereich der Verteilten (Datenbank-) Systeme stammende Techniken zur Replikationsverwaltung oftmals nicht ohne weiteres übertragbar. In der vorliegenden Arbeit untersuchen wir verschiedene bekannte Ansätze zur Konsistenzsicherung von Replikaten bezüglich ihrer Eignung für P2P-Systeme. Darüber hinaus interessiert uns, inwieweit eine Abschwächung der Konsistenzforderung vertretbar ist. Anschließend stellen wir ausgehend von den vorausgegangenen Überlegungen den Entwurf eines Overlaynetzwerkes zur dezentralen Verwaltung von Replikaten vor. Updates erfolgen dabei über einen Masterknoten, welcher sich dynamisch zur Laufzeit ergibt.

## 1 Einleitung

Bereits seit einiger Zeit ist das Peer-to-Peer (P2P) Paradigma Gegenstand der Forschung. P2P-Systeme, d.h. Systeme in denen alle Teilnehmer gleichberechtigt sind, zeichnen sich insbesondere durch ihre erhöhte Skalierbarkeit aus. Durch den Verzicht auf eine zentrale Instanz ergeben sich allerdings auch eine Vielzahl neuer Probleme. So umfassen P2P-Netzwerke oftmals mehrere tausend bis hunderttausend Teilnehmer, welche sich zu beliebigen Zeitpunkten im System ab- bzw. anmelden können. Überdies verwenden die Knoten das Internet als Kommunikationsmedium, sodass es aufgrund von hohen Latenzzeiten oder langsamen Netzwerkanbindungen zu Verzögerungen bei der Datenübertragung kommen kann. Um dennoch auf Anfragen geeignet reagieren zu können, werden die Daten der Teilnehmer auf verschiedene Knoten repliziert. Bei einem Knotenausfall kann anschließend auf entsprechende Replikate zurückgegriffen werden. Außerdem erlauben die Replikate eine effizientere Anfragebearbeitung, so können zum Beispiel komplexe Aufgaben auf mehreren Knoten parallel verarbeitet werden. Liegen Daten in einem System redundant vor, so sind entsprechende Maßnahmen zur Konsistenzsicherung notwendig.

Im folgenden Abschnitt stellen wir bekannte Replikationstechniken aus dem Bereich der verteilten (Datenbank-) Systeme vor und analysieren, inwieweit diese auf die P2P-Problematik übertragbar sind. Darauf aufbauend beschreiben wir in Abschnitt 3 einen eigenen, auf die Besonderheiten von P2P-Umgebungen abgestimmten, Replikationsmechanismus. Die vorgestellten Mechanismen widmen sich dabei ausschließlich der Replikation von Daten und deren Konsistenzsicherung. Es wird nicht darauf eingegangen, wie die Replikate im System wieder aufgefunden werden. Dies bleibt Aufgabe des P2P-Systems (z.B. mittels DHT's).

## 2 Stand der Forschung

**Data Placement**. Datenreplikation hat im wesentlichen zwei Ziele, zum einen die Steigerung der Performance des Systems und zum anderen die Erhöhung der Verfügbarkeit der Daten. In Filesharing-Systemen erfolgt üblicherweise eine implizite Replikation, d.h. Dateien, welche auf einen Peer heruntergeladen werden dienen zeitgleich als Uploadquelle und somit als Kopie. Die Anzahl der Replikate im System ist also direkt abhängig vom Interesse der Nutzer. Daten, die nur selten angefordert werden, verschwinden schneller aus dem System als Daten, die für die Nutzer von großem Interesse sind [BMSV02]. Möchte man die Daten unabhängig vom Grad des Interesses dauerhaft in einem P2P-System verfügbar machen, so sind andere Techniken notwendig. Die einfachste Methode ist das Kopieren der Daten auf eine vorab festgelegte Anzahl von zufällig gewählten Knoten. Das größte Problem hierbei ist die Wartung der Replikate, weswegen man üblicherweise auf die Möglichkeit einer Änderung der Daten verzichtet [SS05b]. Einen anderen Ansatz verfolgen Overlaynetzwerke wie P-Grid [Abe01] und Chord [SMK+01]. Bei diesen wird die logische Struktur des Systems für die Replikation genutzt, sodass auch einfache Wartungsaufgaben realisierbar sind. In [ACMD+03] beschreiben Aberer et al. einen Replikationsmechanismus, welcher die Kopien entlang der dem P-Grid zugundeliegenden Baumstruktur, verteilt. Ein ähnliches Verfahren verwenden Stoica et al. in [SMK+01] zur Erhöhung der Ausfallsicherheit bei Chord. Die Daten werden dabei entlang der virtuellen Ringstruktur kopiert, sodass bei einem Knotenausfall die entsprechenden Daten eventuell noch über die direkten Nachfolgeknoten im Ring verfügbar sind. Beide Methoden ermöglichen Updates lediglich auf einem ausgezeichneten Knoten. Im Allgemeinen entspricht dieser dem Knoten mit der Originalkopie. Betrachtet man die angeführten Techniken so stellt man fest, dass keine bei der Wahl des Replikationspartners auf dessen technische Eigenschaften eingeht. Es stellt sich die Frage, inwieweit es von Nutzen ist, Daten auf einen Knoten zu replizieren, dessen technische Möglichkeiten so gering sind, dass Anfragen nur langsam oder mit Verzögerung bearbeitet werden können bzw. ob es sinnvoll ist Daten auf Knoten zu kopieren, die nur selten oder für kurze Zeit im System verweilen. Insbesondere bei P2P-Systemen, welche auf einer großen Anzahl verschiedenartiger Teilnehmer und einem unzuverlässigen Kommunikationsmedium wie dem Internet basieren, sollte ein Replikationsmechanismus daher bei der Wahl des Replikationspartners auch dessen technische Eigenschaften, wie zum Beispiel Bandweite, Rechenleistung oder Verweildauer, berücksichtigen.

**Synchronisationsart**. Das Verteilen der Updates erfolgt bei den meisten Systemen asynchron (*lazy replication*), d.h. die ursprüngliche Transaktion löst eine Reihe von Subtransaktionen aus, welche die Änderungen zu einem geeigneten Zeitpunkt propagieren [SS05a]. Durch die zeitversetzte Synchronisation kann es passieren, dass verschiedene Versionen eines Datensatzes im System existieren. Im Gegensatz dazu werden bei der *eager replication* alle Kopien im System als Teil der originalen Transaktion synchron aktualisiert. Zum Zeitpunkt der Änderung sind alle Replikate gesperrt, sodass es nicht zu ungewollter Versionierung oder Serialisierbarkeitsproblemen kommen kann. Allerdings steigt die Wahrscheinlichkeit von Deadlocks im System mit zunehmender Knoten- bzw. Transaktionsanzahl erheblich, konkret vertausendfacht sich die Deadlockrate, bei einer Erhöhung der Knotenanzahl von einem auf zehn [GHOS96]. Außerdem wächst die Anzahl der Sperren mit der Anzahl der Knoten, sodass *eager replication* für P2P-Systeme mit mehreren tausend Teilnehmern ungeeignet ist. Beide Synchronisationsarten sind passive Methoden, da die Knoten im System ihre Updates zugewiesen bekommen. Zusätzlich gibt es noch die Möglichkeit, dass Knoten selbst aktiv werden und sich bei anderen Teilnehmern nach Änderungen erkundigen. Meist verwendet man diese Technik, um Knoten, welche das System verlassen haben, mit einem aktuellen Datenbestand wieder einzugliedern [ACMD+03].

**Anzahl der Masterknoten**. Neben der Art der Synchronisation unterscheidet man Replikationssysteme auch nach der Anzahl der schreibberechtigten Knoten. Sind Änderungen nur auf einem ausgezeichneten Knoten möglich, spricht man von einem Single-Master System [GHOS96]. Da dieser spezielle Knoten besonderen Belastungen ausgesetzt ist, wird dafür meist ein leis-

tungsfähiger Teilnehmer gewählt. Fällt er aus, so kommt das komplette System zum Erliegen bzw. es muss durch entsprechende Maßnahmen ein neuer Master bestimmt werden [SL00]. Im Unterschied dazu können beim Multi-Master System auf einer Gruppe von Knoten Änderungen vorgenommen werden. Entspricht die Zahl der Master der Anzahl der Knoten im System so spricht man vom *update anywhere*-Prinzip. Mit Multi-Master Systemen ist es möglich ein Datum gleichzeitig auf verschiedenen Knoten lokal zu ändern, dabei auftretende Inkonsistenzen sollte der Replikationsmechanismus finden und auflösen. Da Änderungen sofort übernommen werden, aber im Falle eines Konfliktes zurückgesetzt werden müssen, befindet sich das System überwiegend in einem inkonsistenten Zustand. Ein anderes Problem von Multi-Master Systemen ist ihre begrenzte Skalierbarkeit aufgrund der hohen Konfliktwahrscheinlichkeit [SS05a]. Beide Systeme scheinen in dieser Form nur wenig geeignet für P2P-Umgebungen. Single-Master Systeme benötigen eine zentrale Instanz und Multi-Master Systeme erreichen in einer dynamischen Umgebung wie P2P-Netzwerken nur schwer einen konsistenten Zustand bzw. skalieren schlecht.

**Replikationsprotokolle**. Gewöhnlich treten Single-Master Systeme in Verbindung mit pessimistischen Replikationsprotokollen auf [BHG87]. Diese Verfahren schränken die Verfügbarkeit der Replikate während einer Änderungsoperation derart ein, dass dem Nutzer der Eindruck vermittelt wird, er besitzt die einzige Kopie der Daten (*single-copy consistency*). Im Gegensatz dazu nimmt man bei optimistischen Protokollen an, dass Updatekonflikte nur selten vorkommen und behandelt diese erst dann, wenn sie wirklich eintreten. Zwar wird hierdurch übermäßiges Sperren vermieden, die von den Multi-Master Systemen bereits bekannten Probleme bleiben aber (auch in diesem Fall) bestehen.

**Propagieren der Updates**. Eine der einfachsten Möglichkeiten Änderungen in einem P2P-Netzwerk zu verbreiten sind epidemische Protokolle [EGKM04]. Diese zeichnen sich insbesondere durch ihre gute Skalierbarkeit und Robustheit aus. Zudem benötigen sie keine speziellen Netzwerktopologien und sind leicht zu entwickeln. Einmal initialisiert ist die Verbreitungen der Nachrichten im Netz allerdings schwer zu stoppen, sodass viele P2P-Systeme Protokolle mit einer zeitlichen Schranke verwenden. Ist diese erreicht, werden die Informationen nicht weiter verbreitet. Typischerweise unterscheiden sich die verschiedenen epidemischen Verfahren dabei lediglich in der Puffergröße der Knoten, deren Fanout und dem Zeitraum, während dessen gepufferte Nachrichten weitergeleitet werden [EGKM04].

**Konsistenzabschwächung**. Bedingt durch die meist zeitversetzte Synchronisation der Replikate existieren in einem P2P-System oft mehre Versionen eines Datums. Es ist somit nicht sichergestellt, dass ein Knoten, welcher eine Anfrage bearbeitet, auch ein aktuelles Datum anbietet. Ist es aus Sicht des Nutzers akzeptabel auf teilweise veraltete Daten zuzugreifen, so kann eventuell kostenintensives Suchen im Netz vermieden werden. Unter der Annahme, dass der Nutzer eine entsprechende Konsistenzabschwächung formulieren kann [GLRG04], ergibt sich dann das Problem, dass Daten einer Transaktion unterschiedliche Gültigkeitszeitpunkte aufweisen können. In [SL03] beschreiben Schlesinger und Lehner eine Methode, welche die Qualität der Konsistenz aktuell verfügbarer Daten untereinander bewertet. Zusätzlich betrachten sie dabei die Änderungsrate eines Datums und kombinieren beide Ergebnisse in einem Inkonsistenzmaß.

Ausgehend von diesen Überlegungen, wollen wir im Folgenden einen Replikationsmechanismus entwerfen, welcher insbesondere auf die Probleme bei P2P-Systemen eingeht. Mit dem vorgestellten System soll es möglich sein, Änderungen an Daten (oder deren Replikaten) von jedem beliebigen Knoten aus vorzunehmen. Updatekonflikte werden dabei weitestgehend vermieden bzw. frühest möglich erkannt und beseitigt. Aus Gründen der Skalierbarkeit und Ausfallsicherheit, soll auf eine zentrale Instanz verzichtet werden.

# 3 Replikationsring

Zur Verwaltung der Replikate konstruieren wir ein Replikations-Overlaynetzwerk basierend auf dem Chord-Protokoll [SMK+01]. Chord ist ein dezentrales Suchprotokoll, welches Schlüssel auf

Knoten abbildet. Das Replikations-Overlaynetzwerk ist dabei unabhängig von der logischen Struktur des zugrunde liegenden P2P-Systems. Der Einfachheit halber nehmen wir im folgenden an, dass ein Replikationsobjekt einem Datum entspricht. Für jedes Datum im System wird nun kontinuierlich ein eigener Chord-Ring (Replikationsring) aufgebaut. Alle Knoten eines Ringes sind dabei gleichberechtigt und erlauben nur lesenden Zugriff auf das entsprechende Datum. Als Schlüssel für den Ring wird ein Rankingmaß $r$, mit $r \in [0, 1]$ eingeführt (siehe Abb. 1). Das Rankingmaß soll Eigenschaften des Knotens, wie zum Beispiel seine Leistungsfähigkeit, Bandweite und mittlere Verweildauer im System, widerspiegeln. Leistungsfähige Systeme, welche häufig verfügbar sind, besitzen ein größeres Rankingmaß als Knoten, die nur selten online bzw. leistungsschwach sind.

Soll auf einem Knoten ein Datum geändert werden, so wird das Update zunächst provisorisch auf einer lokalen Kopie ausgeführt. Anschließend ist für diese Änderung ein Master zu bestimmen, welcher das Propagieren des Datums im Ring übernimmt. Als Master dient der Knoten im Replikationsring, für den gilt $r > succ(r)$. Aufgrund der Konstruktion des Replikationsrings ist dieser Knoten eindeutig bestimmt. *succ(r)* bezeichnet dabei den Peer im Ring, der dem Knoten mit Schlüssel $r$ im Uhrzeigersinn folgt. Kommt es auf dem Master zu einem Konflikt, so sind geeignete Konfliktlösungstrategien notwendig. Andernfalls, können sowohl der Master, als auch der Update auslösende Knoten die geänderten Daten im Replikationsring verteilen. Das Verteilen der Änderungen erfolgt dabei anhand eines epidemischen Verfahrens. Die Knoten, welche ein Update erfahren, werden hierbei zufällig aus den Routingtabellen gewählt. Um zu gewährleisten, dass bei einem Ausfall des momentanen Masters, der nächste Masterknoten ein aktuelles Datum hat, werden die Änderungen zusätzlich an den direkten Vorgängerknoten propagiert.

Durch den Aufbau der Routingtabelle des Chord-Protokolls wird sichergestellt, dass für die Suche des Masters in einem Ring mit $N$ Knoten maximal $\log(N)$ Hops notwendig sind. Aus Performancegründen ist es zudem sinnvoll, einmal erfolgreich identifizierte Master mit in die Routingtabelle aufzunehmen. Die von Chord bekannte Stabilisierungsfunktion kann später unnötige oder falsche Einträge korrigieren.

In dynamischen Umgebungen wie P2P-Systemen ist es üblich, dass Knoten das System zu einem beliebigen Zeitpunkt verlassen bzw. betreten. Ein entsprechender Knoten muss somit eigenständig bei der Wiedereingliederung prüfen, inwieweit sich seine Position im Ring verändert hat und ob Updates verfügbar sind. Darüber hinaus muss er testen, ob seine Position im Ring mit der des aktuellen Masters kollidiert. Ist dies der Fall, so ist der wieder eingegliederte Knoten zu markieren und der gegenwärtige Master davon in Kenntnis zu setzen. Nachdem die letzte Änderung auf diesem Master erfolgreich ausgeführt wurde, muss sie zusätzlich an den wieder eingegliederten Knoten propagiert werden. Außerdem ist der Knoten davon in Kenntnis zu setzen, dass es derzeit keinen Master mehr in diesem Replikationsring gibt.

Ändert sich das Rankingmaß eines Knotens, so wird dieser analog dem Chord-Protokoll aus dem Replikationsring entfernt und anschließend wieder neu im System eingefügt. Um die Wartungskosten dabei gering zu halten, ist es sinnvoll, einen Schwellwert für die Aktualisierung

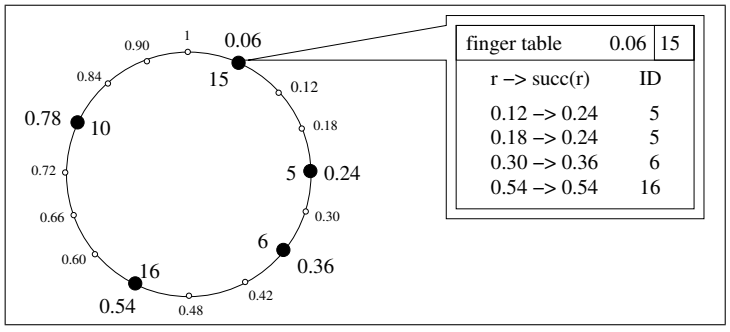


Abbildung 1: Replikationsring und Routingtabelle

des Rankingmaßes einzuführen. Erst wenn dieser überschritten wurde, sollte eine Aktualisierung der Position des Knotens im Replikationsring erfolgen.

Mit dem vorgestellten System ist es möglich, die für eine gewünschte Verfügbarkeit erforderlichen Knoten explizit zu bestimmen [BMSV02] und passende Knoten hinzuzufügen bzw. aus dem System zu entfernen. Im Zusammenhang mit Konsistenzabschwächung, erlaubt die Verwendung des Replikationsrings eine effektive Suche nach zeitlich aktuelleren Daten. Mit jedem Schritt in Richtung Master sollte das Datum zeitlich näher an der aktuellsten Version liegen. Spätestens nach $O(logN)$ Hops ist dann der aktuellste Stand des Replikats erreicht.

# 4 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden bekannte Verfahren aus dem Bereich der Datenreplikation in verteilten Systemen vorgestellt. Darauf aufbauend präsentierten wir einen eigenen dezentral verwalteten Replikationsmechanismus, welcher im Gegensatz zu bestehenden Konstruktionen auch in P2P-Umgebungen Änderungen auf allen Kopien eines Datums erlaubt. Das vorgestellte Overlaynetzwerk arbeitet dabei unabhängig von der logischen Struktur des zugrunde liegenden Systems, sodass Anwendungen auch jenseits von P2P-Netzwerken denkbar sind. Außerdem ermöglicht das Replikations-Overlaynetzwerk die Einbeziehung knotenspezifischer Eigenschaften in den Replikationsprozess. Im weiteren Verlauf unserer Forschungsarbeit werden wir einen ersten Prototypen des vorgestellten Replikationsmechanismus realisieren. Hierfür ist noch ein geeignetes Rankingmaß zu konstruieren.

# Literatur


[Abe01] K. Aberer. A Self-Organizing Access Structure for P2P Information Systems. CoopIS 2001, 2001.

[ACMD+03] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, R. Schmidt, and J. Wu. Advanced peer-to-peer networking: The P-Grid System and its Applications. *PIK Journal - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems*, 2003.

[BHG87] Ph. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[BMSV02] R. Bhagwan, D. Moore, S. Savage, and G. Voelker. Replication strategies for highly available peer-to-peer storage, 2002.

[EGKM04] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, May 2004.

[GHOS96] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD*, pages 173–182, 1996.

[GLRG04] H. Guo, P.-A. Larson, R. Ramakrishnan, and J. Goldstein. Relaxed currency and consistency: how to say "good enough" in sql. In *Proceedings of the 2004 ACM SIGMOD*, pages 815–826, New York, NY, USA, 2004. ACM Press.

[SL00] Y. Saito and H. M. Levy. Optimistic replication for internet data services. In *International Symposium on Distributed Computing*, pages 297–314, 2000.

[SL03] L. Schlesinger and W. Lehner. Konsistenzquantifizierung in Grid-Datenbanksystemen. In *BTW*, pages 404–422, 2003.

[SMK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[SS05a] Y. Saito and M. Shapiro. Optimistic replication. *Computing Surveys*, 37(1):42–81, 2005.

[SS05b] R. Schmidt and G Skobeltsyn. *The P-Grid Documentation*, 2005.