

# Das P-Grid-Overlay-Netzwerk: Von einem einfachen Prinzip zu einem komplexen System

Karl Aberer, Anwitaman Datta, Manfred Hauswirth, Roman Schmidt

School of Computer and Communication Sciences

École Polytechnique Fédérale de Lausanne (EPFL)

1015 Lausanne, Schweiz

## 1 Einleitung

Overlay-Netzwerke haben sich während der vergangenen Jahre zu einem neuen Paradigma im Bereich verteilter Systeme entwickelt. Sie erlauben es durch das Überlagern eines logischen Netzwerkes auf ein physikalisches Netzwerk, die Suche von verteilten Ressourcen unter Verwendung anwendungsspezifischer Schlüssel und Suchkriterien zu unterstützen. Dabei wird auf eine zentrale Koordination weitgehend oder vollständig verzichtet. Popularisiert wurde dieser Ansatz durch Peer-to-Peer-File-Sharing-Systeme basierend auf hierarchischen Overlay-Netzwerken (Napster) oder unstrukturierten Overlay-Netzwerken (Gnutella). Schwachpunkte dieser ersten Generation von Overlay-Netzwerken, wie (teilweise) Zentralisierung oder hoher Bandbreitenbedarf, wurden durch die Entwicklung von *strukturierten Overlay-Netzwerken* behoben. Auf einer logischen Ebene können diese als eine Einbettung eines Graphen (Overlay-Netzwerkes) in einen Schlüsselraum mit einer Distanzfunktion verstanden werden. Durch diese Einbettung werden logische Schlüssel mit physikalischen Adressen verknüpft und logische Suchanfragen können dank einer geschickten Konstruktion des Overlay-Netzwerkes unter Verwendung einer Routingstrategie, die die Topologie des Schlüsselraums ausnützt, effizient ausgeführt werden. Effizienz wird dabei in erster Linie an der verwendeten Bandbreite gemessen.

In diesem Artikel geben wir einen Überblick über P-Grid [Aberer 2001], einer spezifischen Lösung im Bereich der strukturierten Overlay-Netzwerke [Risson et al. 2004], die von uns in den letzten vier Jahren entwickelt wurde. Wir werden dabei weniger auf detaillierte technische Aspekte des Ansatzes eingehen, die in Publikationen bereits ausführlich dokumentiert sind, vielmehr wollen wir einen Überblick über den von einer einfachen Grundidee ausgehenden Entwicklungs-Prozess bis hin zur kompletten System-Implementierung und den dabei auftretenden vielfältigen Problemstellungen vermitteln.

Der Entwurf von P-Grid unterscheidet sich von kanonischen, strukturierten Overlay-Netzwerken, wie beispielsweise Chord, Pastry oder Tapestry [Alima et al. 2004], dadurch, dass er von Beginn an *datenorientiert* war. Ein wesentlicher Aspekt war dabei die Unterstützung nicht-trivialer Suchprädikate, im speziellen von Bereichsabfragen. Eine Voraussetzung dafür stellt die Beibehaltung der Ordnungsrelation von Schlüsseln dar, wodurch randomisiertes Hashing als eine der Standardstrategien für Lastverteilung ausfällt. Konsequenterweise mussten *Lastverteilungsaspekte* von Beginn an beim Entwurf der Overlay-Netzwerk-Struktur und der Algorithmen miteinbezogen werden.

Eine weitere Zielsetzung war es, ein vollständig *dezentralisiertes* und *selbst-organisierendes* System zu entwerfen, um seinen Einsatz unter den typischen Bedingungen, wie sie zum Beispiel bei Peer-to-Peer-File-Sharing-Systemen beobachtet werden, zu ermöglichen. Spezielles Augenmerk wurde daher auf die Robustheit des Systems gelegt, um mit Knoten, die über die Teilnahme am Netzwerk autonom entscheiden, umzugehen. Dies wird einerseits durch Replikation von Daten und Routinginformation erreicht, andererseits unterstützt P-Grid, wie für Overlay-Netzwerke üblich, das dynamische Beitreten und Verlassen eines Netzwerkes („network churn“) und die damit verbundene kontinuierliche Reorganisation. Als Besonderheit haben wir für den effizienten Aufbau von neuen Overlay-Netzwerken außerdem ein parallelisiertes Bootstrapping-Verfahren entwickelt. Dies ist von zentraler Bedeutung sowohl für ein Wiederauflaufen nach katastrophalen Fehlern, als auch für die Nutzung von Overlay-Netzwerken für daten-

orientierte Anwendungen mit sich ändernden Indexierungsanforderungen.

Im Folgenden beschreiben wir wie unsere Entwurfsziele bei der Definition der Netzwerk-Struktur und der auf dieser Struktur operierenden Algorithmen umgesetzt wurden. Dabei wurde besonderer Wert auf die Erlangung analytischer Einsichten in das dynamische Systemverhalten gelegt. Dies wurde durch Modellierung des Systems mit Hilfe von Markov-Modellen umgesetzt. Die dabei gefundenen analytischen Einblicke wurden anhand von Simulationen und einer konkreten Implementierung validiert. Wir gehen des Weiteren auch auf die für die Implementierung zusätzlich zu berücksichtigenden Aspekte ein. Aufbauend auf den Basisdiensten dieser Implementierung werden bereits einige Applikationen im Bereich der semantischen Verarbeitung von Daten (Semantic Web, Information Retrieval) umgesetzt, wodurch wir die Vorteile des datenorientierten Entwurfes demonstrieren können.

## 2 Design

In diesem Abschnitt geben wir einen Überblick über den Entwurf der Struktur von P-Grid sowie den wichtigsten Algorithmen zur Wartung eines P-Grid-Netzwerkes in einer dynamischen Netzwerkumgebung.

### 2.1 Die Struktur von P-Grid

Der Entwurf von P-Grid basiert auf zwei einfachen Prinzipien [Aberer 2001]: 1.) Der Schlüsselraum wird rekursiv unterteilt, sodass jede der resultierenden Partitionen ungefähr die gleiche Anzahl an Schlüsseln enthält und mit ungefähr der gleichen Anzahl Netzwerk-Knoten (im folgenden Peers genannt) assoziiert ist. 2.) Rekursives Unterteilen des Schlüsselraums erzeugt eine kanonische Baum-Struktur, die zur Implementierung eines typischen Präfix-Such-Verfahrens zur effizienten, verteilten Suche verwendet wird. Abbildung 1 stellt diese beiden Ideen graphisch dar.

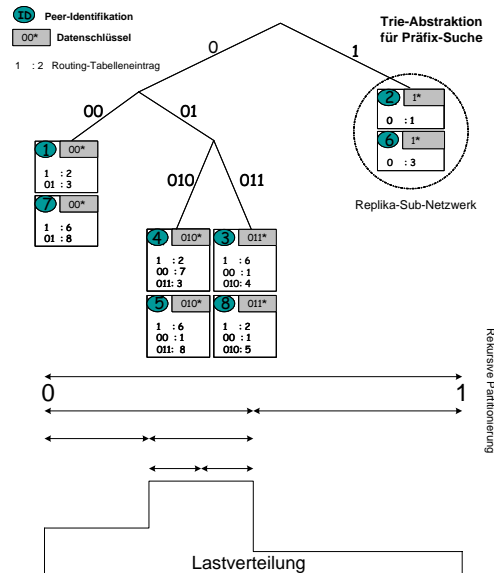


Abbildung 1: Das P-Grid-Overlay-Netzwerk

Wir wählen Datenschlüssel aus einem Intervall, z.B.  $[0, 1)$ . Der untere Teil der Abbildung zeigt eine mögliche unbalancierte Schlüsselverteilung. Das Intervall  $[0, 1)$  wird nun rekursiv unterteilt. Jede Partition wird eindeutig durch einen Bitstring identifiziert, indem bei jeder Partitionierung für das entstehende linke Intervall ein Bit „0“ und für das rechte ein Bit „1“ hinzugefügt wird. Die Partitionierung wird so gewählt, dass die entstehenden Partitionen ungefähr die gleiche Schlüsselanzahl enthalten. Mit jeder Partition wird eine Anzahl Peers (im Beispiel: zwei) assoziiert, und wir bezeichnen den Bitstring der Partition als den

*Pfad* eines Peers. Die Bitstrings aller Partitionen induzieren eine Baum-Struktur<sup>1</sup>, die als Grundlage für ein verteiltes Präfix-Suchverfahren verwendet wird. Jeder Peer verwaltet dazu in einer Routing-Tabelle für jedes Bit seines Pfades Referenzen zu Peers mit einem Pfad, dessen Präfix sich an dieser Bitposition erstmals unterscheidet. Die Suche erfolgt dann durch Bit-weises Auflösen des gesuchten Schlüssels und entsprechendes Weiterleiten anhand der Routing-Tabellen, d.h., wenn ein Bit lokal nicht aufgelöst werden kann wird die Suche an einen für dieses Bit eingetragenen Peer weitergeleitet. Dieses Konstruktionschema wird in der Literatur als strukturiertes Overlay-Netzwerk mit *fixer Unterteilung des Schlüsselraums* [Alima et al. 2004] bezeichnet.

Zur Erhöhung der Ausfallssicherheit bei Knoten- oder Verbindungsausfällen, werden zwei Arten von Replikation verwendet: Zum einen enthält die Routing-Tabelle mehrfache Referenzen für jede Ebene, um so alternative Zugriffswege zu ermöglichen, zum anderen wird versucht immer mehrere Peers mit jeder Datenpartition zu assoziieren, um so Datenredundanz zu unterstützen (*structural replication*).

## 2.2 Effiziente Suche

Im Unterschied zu anderen Overlay-Netzwerken verwendet P-Grid eine variable Pfadlänge und damit eine variable Granularität der Partitionierung des Schlüsselraumes um Lastausgleich zu ermöglichen. Dadurch kann man die Effizienz der Suche nicht mehr strukturell durch eine maximal mögliche Pfadlänge, typischerweise logarithmisch in der Anzahl Peers, sicherstellen. Durch das Partitionierungsverfahren sind in der Tat Pfadlängen und damit Suchkosten linear in der Anzahl der Peers möglich.

Daher muss Sucheffizienz algorithmisch sichergestellt werden: Dazu verwendet P-Grid einen randomisierten Ansatz um Einträge in Routing-Tabellen aus der Menge der Kandidateneinträge zu wählen, im Gegensatz zu dem für Overlay-Netzwerke häufig gewähltem deterministischem Verfahren [Gummadi et al. 2003]. Das heißt beispielsweise, dass Peers, die „0“ als erstes Bit ihres Pfades haben, jeden beliebigen anderen Peer mit „1“ an der ersten Position als Routing-Eintrag auf dieser Ebene verwenden können. Bei einem deterministischen Ansatz hingegen, würde immer ein bestimmter Peer ausgewählt werden, beispielsweise immer der erste Peer im jeweiligen Intervall. Wie wir theoretisch zeigen können, sind die zu erwarteten Routing-Kosten für den randomisierten Ansatz,  $\ln(n)$ , wobei  $n$  die Anzahl der Blattknoten in der dem Overlay-Netzwerk zugrunde liegenden Baumstruktur angibt. Die informelle Erklärung dafür ist, dass durch die zufällige Auswahl die Wahrscheinlichkeit groß ist, mehr als ein Bit in einem Schritt auflösen zu können.

P-Grid's grundlegende Suchfunktionalität wurde bereits auf Bereichsanfragen erweitert. Dies ist in einem Trie relativ einfach möglich, indem zuerst ein für eine Intervallgrenze zuständiger Peer gesucht wird, und dann die Blattknoten bis zur zweiten Intervallgrenze traversiert werden.

## 2.3 Strukturelle Design-Alternativen

Mit dem beschriebenen Entwurf von P-Grid existieren zwei potentielle strukturelle Probleme: 1.) Ein P-Grid-Overlay kann Lastverteilung nicht für beliebig unbalancierte Verteilungen unterstützen. Auf jeder Ebene können  $n$  Peers ein Verhältnis der Lastverteilung von maximal  $1:n-1$  korrekt unterstützen. Dies mag für große  $n$  unkritisch sein, doch auf den unteren Ebenen des Baumes nimmt die Anzahl der mit einer Partition assoziierten Peers sukzessive ab, und damit auch die Möglichkeit Last zu verteilen. 2.) Als eine fundamentale Designentscheidung sind Routing-Tabellen in P-Grid *vollständig*, d.h., auf jeder Ebene existieren zum Pfad des Peers komplementäre Tabelleneinträge. Für sehr unbalancierte Bäume auf Grund sehr schiefer Verteilungen, bedeutet dies, dass die Größe der Routing-Tabellen bis zu linear zur Netzwerkgröße anwachsen kann.

Um diese Probleme zu beheben, wurden zwei alternative Ansätze untersucht. Zum einen besteht die Möglichkeit wenig gefüllte Partitionen nicht mit Peers zu assoziieren, was unvollständige Routing-Tabellen zur Folge hat. Die nötigen Verwaltungsmechanismen sind damit jedoch ungleich komplexer. Alternativ dazu konnten wir kürzlich basierend auf der Theorie von *Small World*-Graphen zeigen, dass optimale Lastverteilung für beliebige Schlüsselverteilungen in randomisierten Overlay-Netzwerken, die eine

---

<sup>1</sup> Genauer gesagt, eine Trie-Struktur. Ein Trie ist eine Baumstruktur mit einem Knoten für jeden gemeinsamen Präfix einer Menge von Strings. Die eigentlichen Daten werden in den Blattknoten abgelegt (Definition nach NIST).

ähnliche Struktur wie P-Grid aufweisen, möglich sind. Die Entwicklung der dazu notwendigen Wartungsmechanismen ist allerdings nicht-trivial und wird noch weitere Entwicklungen benötigen.

Pragmatisch gesehen sind die eingangs genannten Probleme von eher theoretischer Natur. In großen Netzwerken und mit realistischen, selbst sehr schiefen Verteilungen, spielen beide Effekte nur eine untergeordnete Rolle. Nur wenige Peers werden mit Partitionen des Schlüsselraums ohne oder mit nur wenigen Schlüsseln assoziiert und dienen dann dem hauptsächlichsten Zweck der Unterstützung der Such- und Wartungsmechanismen von P-Grid. Aus diesen Gründen wurde für die Implementierung das im Folgenden vorgestellte Design verwendet, das einfache und robuste Such- und Wartungsmechanismen ermöglicht und dennoch einen Grossteil der Applikationen effizient unterstützt.

## 2.4 Dynamisches Verhalten des P-Grid-Overlay-Netzwerks

Aus rein struktureller Sicht basiert P-Grid auf einer verteilten Baum-Struktur, die die Ordnungsrelation der verwendeten Schlüssel beibehält. Damit werden Präfix- und Bereichssuche unterstützt. Um ein Overlay-Netzwerk aufzubauen und zu verwalten sind verteilte Algorithmen notwendig, welche das durch die Struktur vorhandene Potential zur Lastverteilung nutzen und gleichzeitig den Verwaltungsaufwand in dynamischen Netzwerkumgebungen minimieren.

Üblicherweise wird für die Dynamik von Overlay-Netzwerken ein Modell sequentieller Peer-Ein- und -Ausgliederung zu Grunde gelegt. Algorithmen zur Verwaltung eines Netzwerkes bauen für sich eingliedernde Peers neue Routing-Tabellen auf und nehmen Anpassungen bei den bereits teilnehmenden Peers vor. Bei Knotenausfällen oder „unsauberem“ Verlassen müssen Routing-Tabellen repariert werden. Dies wird häufig mittels zeit-periodischer Wartungsalgorithmen durchgeführt. Diese Verfahren können als äquivalent zur Verwaltung einer Datenbank-Indexstruktur verstanden werden.

Zusätzlich zu diesem Modell betrachten wir auch ein in der Literatur bisher weitgehend ignoriertes Problem von grundlegender Bedeutung, nämlich die initiale Konstruktion eines Overlay-Netzwerkes, das so genannte *Bootstrapping*, welches als äquivalent zum Aufbau einer Datenbank-Indexstruktur verstanden werden kann. Die Notwendigkeit dieses Problem zu lösen ergibt sich speziell in datenorientierten Anwendungen, wenn beispielsweise aus sich ändernden semantischen Anforderungen neue Attributtypen von Ressourcen mit Metadatenannotationen effizient zugreifbar gemacht werden sollen. Auch aus Performanzsicht kann es vorteilhaft sein ein Netzwerk periodisch neu zu indizieren anstatt es laufend zu verwalten, z.B. in verteilten Text-Retrieval-Anwendungen. Schließlich kann nach einem katastrophalen Zusammenbruch ein vollständiger Neuaufbau eines Netzwerkes notwendig werden.

Im Folgenden geben wir zuerst einen Überblick über die für P-Grid entwickelte *parallelisierte* Bootstrapping-Strategie und stellen danach die verwendeten Wartungsstrategien für die *sequentielle*, dynamische Ein- und Ausgliederung von Peers dar.

## 2.5 Bootstrapping

Der Bootstrapping-Prozess sollte dezentralisiert sein und möglichst schnell und effizient von statten gehen, d.h., hoch-parallel und mit geringer Bandbreitenbelastung. Gleichzeitig müssen zwei Lastverteilungsaspekte, welche der P-Grid Struktur zugrunde liegen, die Balanciertheit der Anzahl von Schlüsseln pro Partition und der Anzahl von Peers pro Partition, berücksichtigt werden. Um diese beiden Ziele durch einen verteilten Prozess zu erreichen, nützen wir die Tatsache aus, dass die P-Grid-Netzwerkstruktur als das Resultat einer rekursiven Unterteilung des Schlüsselraums verstanden werden kann.

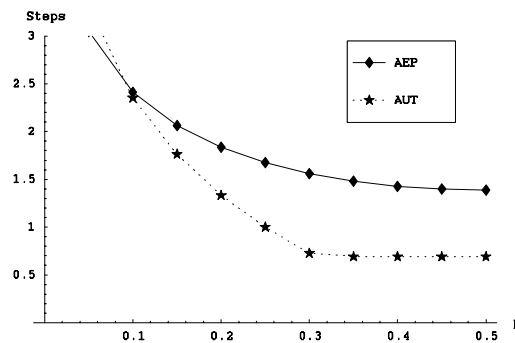
Wir betrachten dazu zuerst den grundlegenden Fall der Aufteilung in zwei Partitionen „0“ und „1“ mit  $n$  Peers, wobei die Aufteilung einer Last von  $p$  bzw.  $1-p$  entsprechen sollte. Dazu entwerfen wir einen verteilten Prozess, in welchem sich Peers zufällig treffen. Dies kann beispielsweise durch *Random Walks* auf einem mit Standardmethoden sehr einfach aufzubauenden unstrukturierten Netzwerk erreicht werden. Bei jedem „Treffen“ entscheiden zwei Peers jeweils lokal, ob sie sich für eine der zwei Partitionen entscheiden sollen. Als Entscheidungsgrundlage verwenden sie dabei den Wert von  $p$ . Global sollte sich die Peer-Population aufgrund der lokalen Entscheidungen gemäß der Lastverteilung im Verhältnis  $p:1-p$  aufteilen. Zusätzlich, und dies macht das Problem nicht-trivial, muss der Prozess garantieren, dass jeder Peer zumindest einen Peer, der sich für die jeweils andere Partition entschieden hat, trifft. Damit haben alle

Peers die notwendige Information, um vollständige Routing-Tabellen aufzubauen.

Zur Partitionierung wurde der folgende, verteilte Algorithmus entwickelt (für  $p \leq 1/2$ ):

1. Jeder Peer, der sich noch nicht entschieden hat, initiiert Interaktionen mit zufällig ausgewählten Peers, bis er eine Entscheidung getroffen hat.
2. Wenn der kontaktierte Peer sich noch nicht entschieden hat, dann führen die Peers mit einer Wahrscheinlichkeit von  $\alpha(p)$  eine balancierte Aufteilung durch und nehmen wechselseitige Referenzen in ihre Routing-Tabellen auf.
3. Hat sich der kontaktierte Peer schon für „1“ entschieden, dann entscheidet sich der Initiator mit einer Wahrscheinlichkeit von  $\beta(p)$  für „0“ und mit einer Wahrscheinlichkeit von  $1-\beta(p)$  für „1“. Im ersten Fall, referenziert der initiiierende Peer den kontaktierten Peer, im zweiten Fall fordert er eine Referenz zur anderen Partition von diesem Peer an.

Dieser Algorithmus wurde als Markov-Prozess modelliert, um mit Hilfe einer transienten Analyse die (nicht-trivialen) Funktionen  $\alpha(p)$  und  $\beta(p)$  zu bestimmen, die die gewünschte Partitionierung des Schlüsselraums bewirken. Zum Beispiel kann  $\beta$  aus der Relation  $p = 1 - \beta^1(1 - 2^\beta)$  ermittelt werden. Mit Hilfe dieses Ansatzes, bezeichnet als *adaptive eager partitioning* (AEP), erzielen wir für realistische Werte von  $p$  deutlich bessere Resultate als mit einem einfachen alternativen Ansatz, den wir als *autonomous partitioning* (AUT) bezeichnen, und welcher darauf beruht, zuerst eine zufällige Entscheidung für eine Partition mit Wahrscheinlichkeit  $p$  zu treffen und dann zufällig andere Peers zu kontaktieren bis ein Peer, der sich für die andere Partition entschieden hat, gefunden wird. Der Unterschied zwischen diesen beiden Ansätzen in Bezug auf die notwendige Anzahl an Interaktionen pro Peer wird in Abbildung 2 dargestellt.



**Abbildung 2: Zur Partitionierung notwendige Anzahl an Schritten (AEP vs. AUT)**

Um diese Methode letztendlich wirklich zur Konstruktion eines P-Grid-Overlay-Netzwerkes einsetzen zu können, sind noch einige weitere nicht-triviale Aufgaben zu lösen: Da der Wert von  $p$  normalerweise nicht bekannt ist, muss er an Hand der den Peers lokal zur Verfügung stehenden Stichproben an Schlüsseln geschätzt werden. Dadurch wird der Konstruktionsprozess mit Fehlern behaftet, die auf nicht-triviale Weise kompensiert werden müssen. Des Weiteren akkumulieren sich Fehler in Bezug auf die Aufteilung auf Grund der Rekursivität des Prozesses. Ebenso muss der Prozess ungefähr synchron ablaufen, um die dem Algorithmus zu Grunde liegenden Annahmen nicht zu verletzen. Der Aufteilungsprozess sollte terminieren, sobald die Anzahl der Peers in einer Partition unter einen bestimmten Grenzwert fällt. Da die Peers jedoch während des Prozesses nicht alle potentiellen Replikate in der gleichen Partition kennen (können), müssen dazu andere Kriterien, basierend auf den lokal verwalteten Schlüsseln, herangezogen werden. Im Zuge ausführlicher Analysen konnten wir Lösungen für alle diese Probleme finden und die mit diesen Lösungen durchgeführten Simulationen zeigen, dass das gewünschte Lastverteilungsverhalten erreicht wird und das Bootstrapping somit effizient durchführbar ist.

## 2.6 Wartung

Nach der Bootstrapping-Phase ändern sich laufend Verteilungseigenschaften, da sich die Last pro Peer und die Anzahl von Peers pro Partition auf Grund der Ein-/Ausgliederung von Peers und dem Einfügen neuer Schlüssel dynamisch ändern. Ähnlich wie beim Bootstrapping verwenden wir probabilistische Prozesse,

um die aktuellen Verteilungseigenschaften zu überwachen und gegebenenfalls durch Verschiebung von Peers zwischen Partitionen bzw. Unterteilung/Vereinigung von Partitionen korrigierend einzugreifen..

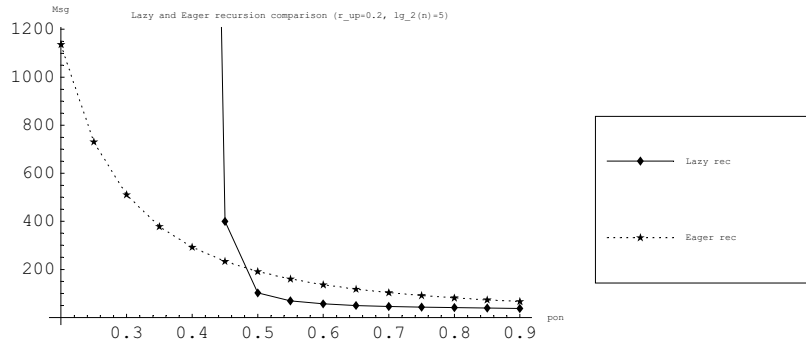
Ein weiterer Aspekt der Wartung ist die Behebung von Inkonsistenzen in Routing-Tabellen, die sich durch temporäre oder permanente Ausfälle von Peers oder Netzwerkverbindungen bzw. durch dynamische Adressvergabe (DHCP) ergeben. Üblicherweise wird versucht diesem Problem durch periodische Suche nach Inkonsistenzen oder gleichzeitig mit Änderungen (*correction on use*) Herr zu werden. Diese Ansätze gehen von einer relativ hohen Verfügbarkeit der Peers aus. Da diese Annahme in typischen Internetumgebungen nur bedingt zulässig ist, gehen wir davon aus, dass Peers sehr häufig nicht verfügbar sind. Aus diesem Grund verwendet P-Grid Routing-Tabellen mit höherer Redundanz, sodass eine hohe Erfolgswahrscheinlichkeit bei der Suche gewährleistet werden kann. Da die Konsistenthaltung einer großen Zahl von Routing-Einträgen zu aufwändig wäre, verwenden wir einen weniger strikten Ansatz und beheben Inkonsistenzen nur wenn eine Suche fehlschlägt [Aberer et al. 2004b]. Dabei unterscheiden wir zwei Strategien: 1.) Als inkonsistent erkannte Tabellen-Einträge werden bei Benutzung korrigiert (*correction on use*) und 2.) erst wenn alle Tabellen-Einträge für eine Ebene ungültig sind, wird repariert (*correction on failure*). Diese weniger strikten Strategien tolerieren einen bestimmten Anteil an ungültigen Einträgen, was den Vorteil hat, dass Peers, die nur temporär nicht verfügbar sind, keinen Reparaturaufwand verursachen. Des Weiteren können Peers ihre Pfade und damit die Verantwortlichkeit für bestimmte Schlüssel auch bei temporärer Nichtteilnahme beibehalten, wodurch der Wartungsaufwand weiter reduziert wird.

Um ungültige Einträge zu reparieren, verwenden wir das Overlay-Netzwerk selbst zur Speicherung der Zuordnungen von eindeutigen logischen Peer-Adressen zu den aktuellen physikalischen. Für die Aktualisierung ihrer aktuellen Zuordnung ist jeder Peer selbst verantwortlich, wobei ein Protokoll eingesetzt wird, um widerrechtliches Ändern durch andere Peers zu verhindern. Für eine Reparatur muss dann im Grunde „nur“ das Overlay-Netzwerk nach der aktuellen Zuordnung abgefragt werden. Dabei kann es natürlich zu weiteren Fehlern auf Grund von Inkonsistenzen kommen, sodass eine Reparatur rekursiv weitere Reparaturen auslösen kann. Dies wirft natürlich die Frage auf, ob ein solches System in einen stabilen Zustand gelangen kann.

Um die Stabilitätseigenschaften unserer Wartungsstrategien zu analysieren, wurde wieder eine Modellierung mit Markov-Prozessen verwendet. In diesem Fall ist allerdings die interessante Frage, unter welchen dynamischen Netzwerkbedingungen sich das Overlay-Netzwerk stabilisiert. Dies erfordert eine Analyse der dynamischen Gleichgewichtseigenschaften des Markov-Prozesses. Für den einfacheren Fall der *correction on use* kann folgende Gleichgewichtsbedingung gegeben werden:

$$p_{on}(1-r_{up})(1-\varepsilon_r)N_{rec} = r_{up}(1-p_{dyn})r \log_2 n,$$

wobei  $p_{on}$  die Online-Wahrscheinlichkeit der Peers,  $r_{up}$  die Änderungsrate der physikalischen Adressen,  $1-r_{up}$  die Abfragerate,  $p_{dyn}$  die Inkonsistenzwahrscheinlichkeit von Routing-Einträgen,  $\varepsilon_r$  die Wahrscheinlichkeit, dass eine Reparatur scheitert,  $N_{rec}$  die zu erwartende Anzahl an zusätzlichen Abfragen pro ursprünglicher Abfrage,  $n$  die Anzahl an Blattknoten und  $r$  die Anzahl an redundanten Routing-Einträgen pro Ebene angeben. Die linke Seite der Gleichung repräsentiert die durch Suchanfragen reparierte Anzahl an Routing-Einträgen, die rechte Seite hingegen die gleichzeitig durch Änderungen im Netzwerk ungültig werdenden Einträge. Ist das System in einem dynamischen Gleichgewicht, dann hält sich die Anzahl der Änderungen mit der Anzahl der Reparaturen die Waage. Um diese Gleichungen zu lösen, können aus den strukturellen Eigenschaften eines P-Grid Netzwerkes weitere Beziehungen zwischen  $N_{rec}$ ,  $\varepsilon_r$  und  $p_{dyn}$  hergeleitet werden. Die Auswertung der Modelle liefert Einsichten in die Eigenschaften der Strategien. Abbildung 3 zeigt den Nachrichten-Overhead (Msg) für *correction on failure* bzw. *correction on use* für unterschiedliche Netzwerkumgebungen mit unterschiedlichem  $p_{on}$  und bei einer konstanten Änderungsrate von  $r_{up}=0.2$ .



**Abbildung 3: Sofortige Reparatur (*correction on use*) vs. Korrektur bei Ausfall (*correction on failure*)**

Es zeigt sich, dass bei schlechter Verlässlichkeit des Netzwerks (niedriges  $p_{on}$ ), *correction on use* besser abschneidet als *correction on failure*. Dies erklärt sich daraus, dass sich bei *correction on failure* fehlerhafte Routing-Einträge akkumulieren, bis eine Korrektur nicht mehr möglich ist (*thrashing*). Wenn andererseits das Netzwerk eher stabil ist, ist der Durchsatz von *correction on failure* besser, da nicht fortwährend Aufwand für Reparaturen notwendig wird und sich Fehler erst akkumulieren müssen, bevor repariert wird. Mit Simulationen konnten wir verifizieren, dass unser kontinuierliches, analytisches Modell das Verhalten des diskreten Systems für ausreichend große Peer-Populationen (einige hundert) recht genau beschreibt.

Weiters bietet P-Grid auch noch die Möglichkeit von Updates der von Peers gespeicherten Datenschlüssel. Da Peers eine hohe Anzahl an Replikaten besitzen können, kann der durch die Weiterleitung von Änderungen notwendige Aufwand beträchtlich sein. Um unnötige Update-Nachrichten zu vermeiden, verwendet P-Grid ein randomisiertes Schema basierend auf *Gossiping*.

### 3 Implementierung

Grundlage der Entwicklungsphilosophie von P-Grid ist es, die theoretischen Konzepte laufend in einer konkreten Implementierung zu validieren. Zentrale Fragen bei der Implementierung waren das Design der Kommunikationsprotokolle, die Entwicklung einer flexiblen, erweiterbaren Softwarearchitektur und der praktische Test von großen Netzwerken.

#### 3.1 Protokolle

Bei der Spezifikation der verteilten Algorithmen wurde angenommen, dass Peers bilateral durch einen gemeinsamen Zustandsraum interagieren. In einer Implementierung muss dies auf nachrichten-basierte Protokolle abgebildet werden. Dazu wurden Protokolle zur Teilnahme an einem Overlay-Netzwerk, zur Suche und zur Netzwerkverwaltung entwickelt. Eines der Ziele war es, dabei die Bandbreite für den Austausch von Zustandsinformationen der Peers zu minimieren. Wir beschreiben dies am Beispiel des beim Bootstrapping verwendeten Protokollablaufs.

Jeder Peer unterhält eine Liste von zufällig ausgewählten Peers im Netzwerk (analog zu Gnutella) um zufällige Interaktionen zu initiieren. Das Protokoll startet damit, dass ein Peer eine Einladung, die nur den eigenen Pfad enthält, an einen dieser zufällig ausgewählten Peers schickt. Der kontaktierte Peer kann diese Einladung annehmen oder ablehnen, d.h. nicht reagieren. Nimmt er an, antwortet er indem er dem Einlader eine Nachricht schickt, die seine Routing-Tabelle, und alle von ihm gespeicherten Datenschlüssel, die mit dem Pfad des Einladenden übereinstimmen, enthält. Danach wartet er auf eine ebensolche Nachricht als Antwort. Wird der Austausch dieser Informationen erfolgreich abgeschlossen, können beide Peers unabhängig voneinander den Partitionierungsalgorithmus basierend auf einer konsistenten Sicht des wechselseitigen Zustands fortsetzen und ihre Pfade und Daten entsprechend anpassen.

Um größtmögliche Portabilität zu gewährleisten, wurde ein auf XML basierendes Nachrichtenformat gewählt [P-Grid 2004]. Abbildung 4 zeigt als Beispiel eine Suchanfrage in der ein Peer, identifiziert durch seine global eindeutige Kennung (GUID) und via Adresse/Port erreichbar, nach Dateien sucht, welche

„Load balancing“ im Dateinamen enthalten. Suchanfragen selbst enthalten ebenfalls eine global eindeutige Kennung und sind immer für einen bestimmten Informationstyp bestimmt. Zum Beispiel wird der Typ „text/file“ zur Suche nach Dateien verwendet. Der binäre Schlüssel (Key) einer Suchanfrage wird mit Hilfe einer präfix-erhaltenden Hash-Funktion aus dem Schlüsselwort erstellt und dient zum Auffinden des für diese Suchanfrage verantwortlichen Peers, d.h. eines Peers, dessen Pfad ein Präfix des Schlüssels ist. Im Feld Index wird dabei der Fortschritt der Suche protokolliert (Länge des übereinstimmenden Präfix zwischen Suchanfrage und Pfad in Bits).

```

<P-Grid Version="1.0" Content-Length="237">
  <Host GUID="ADDA0D1EE6A4C93C79E56318245F05450C70B0"
    Address="pc2.epfl.ch" Port="1805"/>
  <Query GUID="6605B37EA38F29ABF4C48A71A0CC5CCFF03207"
    Type="text/file" Index="0" Key="10000010101001">
    <![CDATA[Load balancing]]></Query>
</P-Grid>

```

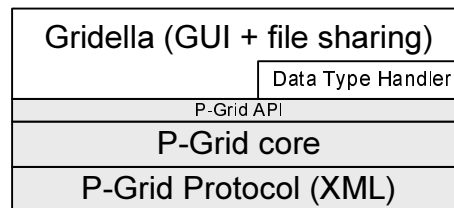
**Abbildung 4: Eine P-Grid-Suchanfrage**

Die Verwendung von XML führt zu größeren Nachrichten und damit mehr Bandbreitenverbrauch. Daher verwenden wir Kompression (ZLIB), wodurch der Bandbreitenverbrauch auf ca. 30-40% der XML-Nachrichtengröße gesenkt werden kann. Dies erhöht zwar den Verbrauch an lokalen Ressourcen (CPU), jedoch ist in Overlay-Netzwerken Bandbreite die dominierende Ressource. Die Größe der Protokollnachrichten reicht von wenigen Bytes (Anfrage) bis hunderten kB für Partitionierung und Anfrage-Resultate, abhängig von der Anzahl der von einem Peer verwalteten Schlüssel bzw. der Größe des Suchergebnisses.

Für den Aufbau und die Wartung eines P-Grid-Overlay-Netzwerkes werden unter Umständen häufig Zustandsinformationen ausgetauscht, die abhängig von der Menge der verwalteten Schlüssel beträchtliche Größe erreichen können. Andererseits sind die ausgetauschten Schlüsselmengen häufig recht ähnlich. Um daher den Übertragungsaufwand weiter zu minimieren, verwendet P-Grid algebraische Signaturen, um durch Erkennung identischer Schlüsselmengen unnötiges Übertragen zu verhindern.

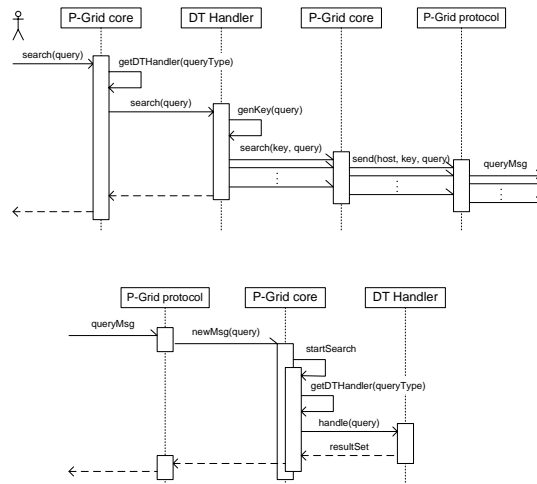
### 3.2 Softwarearchitektur

Abbildung 5 zeigt die geschichtete Architektur der aus den zwei Hauptschichten P-Grid-Bibliothek (grau) und Gridella (File-Sharing als Beispielapplikation) bestehenden Implementierung von P-Grid.



**Abbildung 5: Softwarearchitektur von P-Grid**

Die P-Grid-Bibliothek stellt alle für die Konstruktion und die Wartung eines P-Grid-Overlay-Netzwerkes notwendigen Funktionalitäten sowie die Suchfunktionalität zur Verfügung. Die darüber liegende Applikationsschicht verwendet diese Funktionalitäten über das P-Grid-API und kann die P-Grid-Funktionalitäten durch Adaptionenklassen (*Data Type Handlers*) an anwendungsspezifische Anforderungen anpassen, wobei Anwendungen beliebig viele dieser Adaptionenklassen verwenden können. Diese Adaptionenklassen legen unter anderem fest, wie die zur Indexierung verwendeten Schlüssel aus den zu indexierenden Daten generiert werden und wie die lokale Datenbank des Peers durchsucht wird. Das P-Grid-API bietet unter anderem Funktionen für das Einfügen, Verändern, Löschen und Auffinden von Daten (*insert(data)*, *update(oldData, newData)*, *delete(data)*, *search(query)*). Abbildung 6 zeigt den Ablauf einer Suchoperation im Kontext dieser Architektur.



**Abbildung 6: Ablauf einer Suchoperation**

Die Applikation startet eine Suchoperation durch den Aufruf einer API-Funktion, der die Suchkriterien übergeben werden. Der P-Grid-Kern ermittelt die zuständigen applikationsspezifischen Anpassungsfunktionen, übergibt diesen die Suchanfrage zur Umsetzung in die für die Suche zu verwendenden Schlüssel (wieder mit Hilfe von vom Kern zur Verfügung gestellten Funktionalitäten) und überprüft, ob der Peer die Anfrage selbst beantworten kann. Ist dies der Fall, durchsucht der Data Type Handler die lokale Datenbank und liefert die gefundenen Treffer. Ist der lokale Peer nicht zuständig, wird asynchron die Anfrage vom Kern an einen anderen Peer über die Protokollschicht weitergeleitet (entsprechend dem P-Grid-Suchalgorithmus). Die P-Grid-Bibliothek ist als Middleware für Entwickler ausgelegt und mit vielen Anwendungsbeispielen und Java-API-Dokumentation dokumentiert.

## 4 Experimente

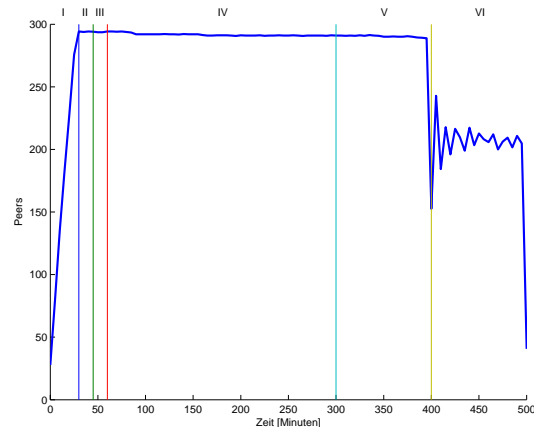
Um experimentelle Resultate mit nicht-trivialen Netzwerkgrößen unter realistischen dynamischen Netzwerkbedingungen zu erhalten, wurde P-Grid auf der PlanetLab-Infrastruktur (<http://www.planet-lab.org/>) getestet. PlanetLab [Chun et al. 2003] ist eine globale Testumgebung um verteilte Applikationen im großen Rahmen unter realistischen Bedingungen zu testen. Derzeit besteht PlanetLab aus ca. 530 über den Erdball verteilten Knoten. Als Betriebssystem wird eine modifizierte Linux-Version basierend auf RedHat Fedora Core 2 verwendet, um die effiziente und sichere Administration von PlanetLab zu ermöglichen, ein einfaches Setup der Experimente zu gewährleisten und den Ressourcenverbrauch der Experimente unter Kontrolle zu halten. Die P-Grid-Experimente auf PlanetLab wurden mit ca. 300 Knoten, abhängig von der Anzahl an verfügbaren Knoten, durchgeführt, wobei auf jedem Knoten eine P-Grid-Instanz lief.

Mit Hilfe der Experimente auf PlanetLab wollten wir verifizieren, inwiefern das tatsächliche Systemverhalten mit den theoretischen Vorhersagen übereinstimmt. Im Folgenden präsentieren wir einige der dabei erhaltenen Resultate. Bei deren Interpretation muss bedacht werden, dass PlanetLab von vielen Forschungsgruppen auf der ganzen Welt verwendet wird und die vielen gleichzeitigen Experimente sich daher gegenseitig beeinflussen.

Das Setup der Experimente war wie folgt: Nach Verteilung des Programmcodes auf die Knoten werden den Peers Schlüssel zugeteilt, welche gemäß unterschiedlicher Verteilungen generiert werden. In einer Anfangsphase (Phase I, 30 Minuten) kontaktieren die Peers in zufälliger Reihenfolge einen ausgewählten Peer, und formen danach ein unstrukturiertes Netzwerk in dem während 15 Minuten (Phase II) Tabellen mit zufällig ausgewählten Peers ausgetauscht werden, um einen realistischen Anfangszustand mit einem randomisierten, unstrukturierten Overlay-Netzwerk zu erhalten. In der nächsten Phase (Phase III) erzeugt jeder Peer 5 Replikate seiner Schlüssel auf anderen Peers (15 Minuten), um die Verfügbarkeit zu erhöhen. Danach wird der Bootstrapping-Prozess gestartet. In dieser Phase (Phase IV, 240 Minuten) interessiert uns besonders der Bandbreitenverbrauch und ob die theoretisch vorhergesagten Lastverteilungseigenschaften

unter realistischen Bedingungen erreicht werden. Nach insgesamt 300 Minuten wird dann das entstandene Overlay-Netzwerk für Experimente zur Feststellung der Sucheffizienz verwendet (Phase V), wobei jeder Peer alle 1-2 Minuten eine Suche durchführt. Danach wird Netzwerkdynamik simuliert (Phase VI), um die Robustheit des Suchalgorithmus zu testen. Dabei gehen Peers alle 5-10 Minuten für 1-5 Minuten offline.

Abbildung 7 zeigt die Anzahl der zu bestimmten Zeitpunkten am Overlay teilnehmenden Peers. Die Abbildung zeigt wie Peers sukzessive am Overlay-Netzwerk teilnehmen, und danach ihre Anzahl (ca. 300) in der Bootstrapping-Phase konstant bleibt. Da in Phase VI Ausfälle simuliert werden, nimmt die Anzahl substantiell ab auf ca. 220 Peers. Am Ende des Experiments verlassen die Peers das Netzwerk wieder (daher die rapide sinkende Anzahl am Ende von Phase VI):



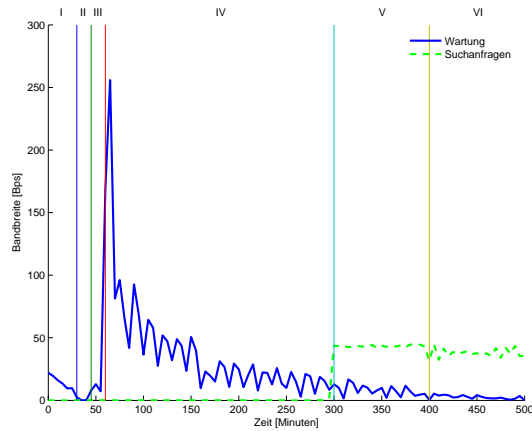
**Abbildung 7: Anzahl teilnehmender Peers**

Aufgrund des beträchtlichen Aufwands, der notwendig ist, um die Experimente aufzusetzen und auszuwerten, der langen Laufzeit der Experimente, sowie der zeitintensiven Analyse zum Beheben von Programmfehlern in einem verteilten System, ist das Erstellen statistisch relevanter Ergebnisse noch Teil der laufenden Arbeit. Allerdings lassen sich aus einzelnen Experimenten schon relevante Rückschlüsse auf die grundlegende Funktionsfähigkeit des Systems ziehen.

Im Folgenden beschreiben wir einige der aus einem spezifischen Experiment erhaltenen Resultate. Dabei generierten wir Schlüssel aus einer Dokumentensammlung für ein Text-Retrieval-Experiment. Jedem Peer wurden zufällig 10 Schlüssel zugeteilt. Die geringe Schlüsselzahl wurde gewählt, um die Durchführung der Experimente mit geringerem Zeitaufwand bewältigen zu können. Es wurden zur Verifizierung allerdings auch vergleichbare Experimente mit bis zu 2000 Schlüsseln pro Peer durchgeführt.

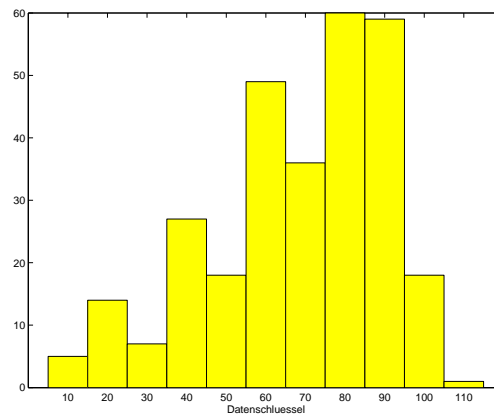
Abbildung 8 zeigt den globalen Bandbreitenverbrauch aller Peers, der sowohl aus der Wartung als auch aus Suchanfragen resultiert (in Bytes/s gemittelt über eine Minute). Wie man sieht, steigt der Verbrauch kurzzeitig auf 250 Bytes/s pro Peer am Beginn des Bootstrapping. Dies erklärt sich dadurch, dass zu Beginn zwischen zwei Peers alle Schlüssel und die Routing-Tabellen ausgetauscht werden. Der theoretisch erwartete Bandbreitenverbrauch für eine Interaktion pro Peer in einer Minute liegt bei ca. 200 Bytes/s (ohne Protokoll-Overhead). Die Wartungskosten nehmen während des Bootstrapping stetig ab und werden schließlich vernachlässigbar niedrig im Vergleich zum Aufwand für Suchanfragen. Dies resultiert aus dem Rückgang der ausgeführten Interaktionen zwischen Peers als auch aus dem Rückgang der Anzahl der zu übertragenden Schlüssel. Peers entscheiden autonom über die Anzahl der notwendigen Interaktionen und über die von einer Interaktion betroffenen Schlüssel, basierend auf dem Pfad des jeweiligen Interaktionspartners.

Die Suchkosten in Abbildung 8 resultieren aus den im Experiment initialisierten Suchanfragen (ca. alle 1-2 Minuten pro Peer). Jede Suche liefert dabei genau einen Treffer. Damit ergibt sich eine untere Schranke für den Bandbreitenverbrauch (natürlich abhängig von der Suchfrequenz). Der Suchaufwand reduziert sich in der letzten Phase des Experiments leicht auf Grund der geringeren Anzahl an teilnehmenden Peers, welche Suchabfragen stellen.



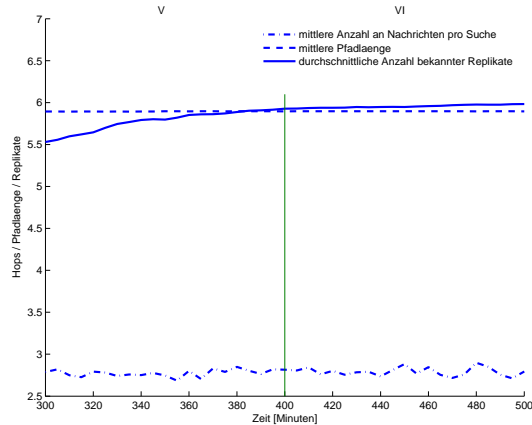
**Abbildung 8: Bandbreitenverbrauch**

Abbildung 9 zeigt anhand eines Histogramms die Lastverteilung der Schlüssel unter den Peers, die eng um den Durchschnitt konzentriert ist. Eine kleinere Anzahl an Peers ist unterlastet bzw. überlastet, was bei der gewählten Partitionierungsstrategie zu erwarten ist.



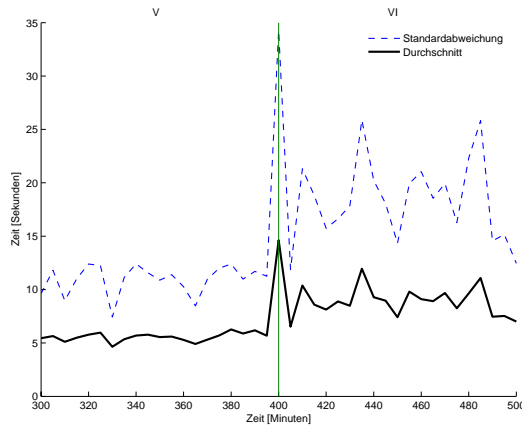
**Abbildung 9: Lastverteilung (Daten)**

Abbildung 10 fasst weitere Aspekte des Overlay-Netzwerkes zusammen. Zum ersten sieht man, dass die für die Beantwortung von Suchanfragen notwendigen Netzwerknachrichten (*Hops*) nach Aufbau des Overlay auf einem erwarteten niedrigen Niveau bleiben und dies größenordnungsmäßig nur die Hälfte der durchschnittlichen Pfadlänge ausmacht. Dies resultiert aus der zufälligen Auswahl eines Peers aus den möglichen Kandidaten für den nächsten „Hop“ während der Suche, was mit großer Wahrscheinlichkeit dazu führt, dass mehr als ein Bit in einem Schritt aufgelöst wird. Weiters zeigt sich, dass nach vollständigem Netzwerkaufbau, was sich in der sich stabilisierten durchschnittlichen Pfadlänge widerspiegelt, die Anzahl der Replikate pro Peer über dem erwarteten Wert von 5 Replikaten pro Peer liegt. Die Anzahl der bekannten Peers steigt am Beginn der Suchphase noch leicht, da während der Bootstrapping-Phase noch nicht alle Peers alle Replikate „entdeckt“ haben.



**Abbildung 10: Mittlere Anzahl an Nachrichten pro Suche, mittlere Pfadlänge und durchschnittliche Anzahl bekannter Replikate**

Abschließend zeigt Abbildung 11 die durchschnittliche Latenz und deren Standardabweichung bei Suchoperationen. Die absoluten Werte sind relativ hoch und spiegeln das schlechte Antwortzeitverhalten der PlanetLab-Knoten wider, das aus der sehr hohen Belastung der Infrastruktur durch parallel durchgeführte Experimente resultiert. Die Antwortzeiten sind in der zweiten Phase aufgrund der simulierten Ausfälle leicht erhöht.



**Abbildung 11: Antwortzeiten für Suchoperationen**

## 5 Anwendungen

Durch die Verfügbarkeit einer lauffähigen Implementierung wird P-Grid bereits für die Realisierung diverser Applikationen verwendet, die wir im Folgenden kurz vorstellen.

### 5.1 Content Sharing

Unsere Demoapplikation Gridella unterstützt das von anderen Peer-to-Peer-Systemen her bekannte *Content Sharing*, wobei Gridella Volltext-Suche auf den Namen der im System befindlichen Dateien ermöglicht. Volltextsuche wurde mittels Indizierung aller Suffixe der Dateinamen implementiert. Suchergebnisse werden in einem GUI angezeigt und die gefundenen Dateien können heruntergeladen werden. Des Weiteren bietet das GUI Statistiken über den Dateitransfer (beide Richtungen), ermöglicht es die lokale Bibliothek an Dateien zu verwalten, und liefert Informationen über den Zustand und den Aufbau des P-

Grid-Netzwerkes. Das GUI ist in sieben Sprachen verfügbar und ermöglicht es, alle Systemeinstellungen komfortabel vorzunehmen. Der Quellcode sowie der Programmcode von P-Grid und Gridella für diverse Plattformen sind unter <http://www.p-grid.org/> verfügbar.

## 5.2 Semantische Annotationen

Um semantische Suche unterstützen zu können, wurde Gridella zu GridVine erweitert [Aberer et al 2004a], welches semantische Annotationen von Inhalten in RDF unterstützt. Peers können ihre Daten lokal annotieren und dabei lokale Annotationschemata verwenden. Zusätzlich können Peers Abbildungen ihres eigenen Schemas auf andere Schemata zur Verfügung stellen. Das P-Grid-Overlay-Netzwerk wird dann als Repository für Schemata und Abbildungen und für die Durchführung von strukturierten Suchoperationen verwendet. Bei Suchoperationen können passende Abbildungen gefunden werden, die es ermöglichen Suchanfragen zu transformieren und an andere semantische Domänen weiterzuleiten. Wir bezeichnen diesen Ansatz als *Semantic Gossiping*.

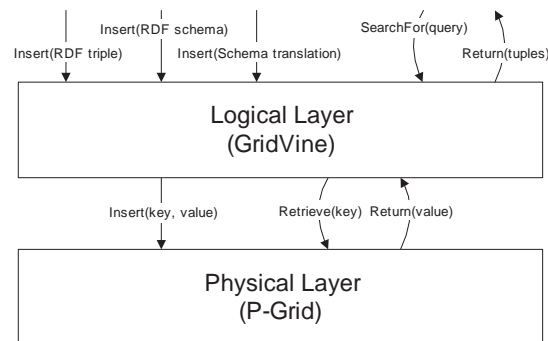


Abbildung 12: Architektur von GridVine

## 5.3 Peer-to-Peer-Retrieval

P-Grid wird als Infrastruktur verwendet, um invertierte Dateien verteilt zu verwalten und damit verteiltes Volltext-Retrieval zu unterstützen. Das Hauptproblem bei sehr großen Dokumentenmengen liegt dabei darin, Schlüssel so zu selektieren, dass die *Posting Lists* für indizierte Schlüssel nicht zu schnell anwachsen und damit substantielle Netzwerkbelastung durch den Zugriff auf die verteilten invertierten Dateien verursacht würde. Wir erreichen dies durch Vorverarbeitung der Vokabulare, bei der seltene Termengenen als Schlüssel ausgewählt werden. Allgemeine Anfragen werden aufgrund statistischer Eigenschaften der Textkollektion auf die selektierten Schlüssel abgebildet. Damit konnte die Netzwerkbelastung um eine Größenordnung reduziert werden, ohne dafür die Qualität der Suche zu beeinträchtigen. Damit haben wir einen entscheidenden Schritt in Richtung der Verwendung eines strukturierten Overlay-Netzwerkes für Volltext-Indizierung gemacht.

## 6 Ausblick

Dieser Artikel gab einen Überblick über das Design und die Implementierung von P-Grid, eines datenorientierten, dezentralisierten, strukturierten Overlay-Netzwerkes. Wir konnten zeigen, dass es möglich ist strukturierte Overlay-Netzwerke stabil zu implementieren und die theoretisch vorhergesagten Eigenschaften der entwickelten randomisierten Algorithmen in Experimenten zu erreichen, auch wenn diese unter stark abstrahierenden Annahmen hergeleitet wurden. Die absoluten Performanzwerte müssen dabei mit einer gewissen Vorsicht interpretiert werden, da PlanetLab eine relativ künstliche Testumgebung ist und wir andererseits viele mögliche Optimierungen nicht verwendet haben. In den vorgestellten Experimenten haben wir eine gewisse Kontrolle bei der Wahl der Systemparameter und bei der Synchronisation des Ablaufs der Netzwerk-Evolution ausgeübt. Wie damit bei einem konkreten Einsatz des Systems umgegangen wird, führt unmittelbar zur grundlegenden Frage der verteilten Administration von

unterschiedlichen Anwendungen in Overlay-Netzwerken, die sowohl weitere Forschungsarbeiten als auch praktische Evaluierungen erfordert.

## 7 Literatur

[Aberer 2001] Aberer, K.: P-Grid: A self-organizing access structure for P2P information systems. In: Lecture Notes in Computer Science 2172, 2001, S. 179-194.

[Aberer et al. 2004a] Aberer, K.; Cudré-Mauroux, P.; Hauswirth, M.; van Pelt, T.: GridVine: Building Internet-Scale Semantic Overlay Networks. In: Proceedings of the 3rd International Semantic Web Conference (ISWC04), 2004, Hiroshima, Japan.

[Aberer et al. 2004b] Aberer, K.; Datta, A.; Hauswirth, M.: Efficient, self-contained handling of identity in Peer-to-Peer systems. In: IEEE Transactions on Knowledge and Data Engineering, 16. Jg., 2004, Heft 7, S. 858-869.

[Alima et al. 2004] Alima, L.O.; Ghodsi, A.; Haridi, S.: A Framework for Structured Peer-to-Peer Overlay Networks. In: Post-proceedings of the Global Computing Conference, Springer-Verlag LNCS, 2004.

[Chun et al. 2003] Chun, B.; Culler, D.; Roscoe, T.; Bavier, A.; Peterson, L.; Wawrzoniak, M.; Bowman, M.: PlanetLab: An Overlay Testbed for Broad-Coverage Services. In: ACM SIGCOMM Computer Communication Review, 33. Jg., 2003, Heft 3.

[Gummadi et al. 2003] Gummadi, K.; Gummadi, R.; Ratnasamy, S.; Shenker, S.; Stoica, I.: The Impact of DHT Routing Geometry on Resilience and Proximity. In: ACM SIGCOMM, 2003.

[P-Grid 2004] P-Grid Protocol Version 1.0, 2004. <http://www.p-grid.org/specs/protocol.html>

[Risson et al. 2004] Risson, J.; Moors, T.: Survey of research towards robust peer-to-peer networks: search methods. In: Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia, September 2004.