-Seminararbeit-

Datenumschichtung auf Parallelen Datenträgern

Roman Matthias Keil

Algorithmen und Komplexität
Institut für Informatik
Fakultät für Angewandte Wissenschaften
Albert Ludwigs Universität Freiburg

28. Juli 2008

Prinzip Storage-Area-Network

Bereits in der Studie "How much information?" [1] zeigte uns die "University of California" den drastischen Zuwachs an Speicherbedarf. Alleine im Jahr 2002 wurden ungefähr 5 Trillionen[1] neue Daten produziert und das weltweit gespeicherte Datenvolumen hat sich in den Jahren 1999 bis 2002 nahezu verdoppelt. Wir leben in einer Zeit der Datenexplosion, wo jedes Jahr mehr Daten hinzukommen und den weltweiten Bedarf an Speicherplatz drastisch in die Höhe treibt. Durch den immensen Anstieg der Datenmenge steigt nicht nur die Anforderung an mehr Speicherplatz, sondern auch an Geschwindigkeit und Datenverfügbarkeit.

Einzelne Rechner bewältigen diese Aufgabe schon lange nicht mehr, stattdessen werden Daten, ähnlich wie bei dem für physikalische Festplatten[2] bekannten RAID-System[3], in Speichernetzwerken (Storage-Area-Network, kurz: SAN) abgelegt.

RAID-System

Einzelne Festplatten in Rechner sind langsam und im Falle eines Defekts sind die Daten nur schwer rekonstruierbar. Zur Verbesserung schließen wir einzelne Festplatten zu einem Festlattenverbund zusammen, welcher es uns ermöglicht die Daten mittels Segmentierung[4] und Redundanz[5] schneller und sicherer abzulegen. Nun speichern wir zwar unsere Daten im Rechner effizienter, allerding entstehen neue Engstellen, der Rechner selbst und die Netzwerkverbindung.

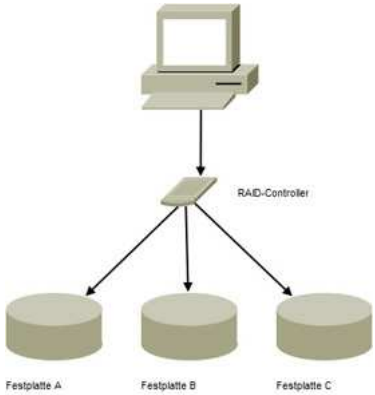


Abbildung 1: Aufbau eines RAID-Systems

Storage-Area-Network

Speichern wir Daten nur auf einem Rechner muss dieser bei hoher Dateianfrage die Netzwerklast alleine bewältigen, was dazu führt das Teile in unserem Netzwerk ausgelastet sind, und andere noch Kapazitäten zur Verfügung haben. Auch haben wir

[1] eine Trillion entspricht (10^{18} oder ca. 2^{60}) Bytes
[2] unter einer Physikalischen Festplatte versteht man ein, in einen Rechner, fest integriertes Speichermedium
[3] redundant array of independent disks - dient der Verknüpfung mehrere physikalischer Festplatten eines Computers zu einem logischen Laufwerk
[4] aufteilen oder zerteilen in verschieden Teilstücke
[5] bedeutet mehrfach oder im Überfluss vorhanden

im Falle, dass unser Rechner ausfällt, keine Möglichkeit an die Daten zu kommen. Anstatt nun eine Datei nur auf einem Rechner zu speichern, können wir das RAID-Prinzip übernehmen und auf unsere Rechner übertragen. Die einzelnen Clientsysteme eines Speichernetzwerks werden nun nicht mehr direkt angesprochen, sondern der Datentransfer geht über einen zentralen SAN-Server, welcher auf die Daten, identisch dem RAID-Controller, Segmentierung und Redundanz anwendet und die Daten an die SAN-Clientsysteme verteilt. Wir erreichen Damit eine bessere Ausnutzung der uns zur Verfügung stehenden Bandbreite im Netzwerk und damit eine Absicherung gegen Anfragespitzen so wie die Möglichkeit unsere Daten, auch bei Ausfall eines Rechners, zu rekonstruieren.

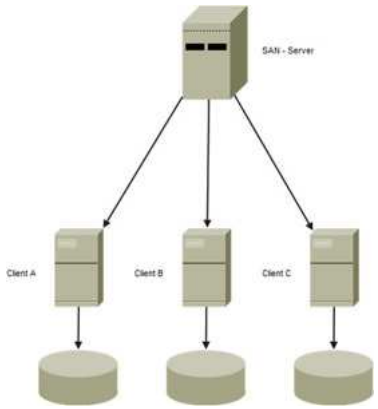


Abbildung 2: Modell eines Storage-Area-Network

Datenverteilung im Storage-Area-Network's

Wir stellen uns nun die Frage, wie, anhand welcher Kriterien, sollen wir unsere Daten im Speichernetzwerk verteilen. Wir wollen unsere Daten einerseits gegen Datenverlust absichern, und andererseits möglichst Speicherplatzeffizient[6] ablegen. Auch wollen wir unser Netzwerk möglichst effizient nutzen um eine möglichst höhe Übertragungsgeschwindigkeit zu erreichen. Dazu wollen wir Daten, welche sehr oft angefragt werden, auf schnellen Rechnern mit schnellen Netzwerkverbindungen ablegen.

All dies ähnelt einem uns aus der theoretischen Informatik bekannten Problemstellung, dem Rucksackproblem[7]. Wir können hier jedes unserer Clientsysteme als eigenen Rucksack betrachten. Leider ist damit die Berechnung einer optimalen Platzierung unserer Daten im Speichernetzwerk NP-Vollständig. Zur Berechnung einer optimalen Datenverteilung werden daher meist Approximationsalgorithmen, wie der in [3] beschriebene "Sliding-Window-Algotithmus", verwendet.

Problemstellung

Eines der größten Probleme, welches wir hierbei haben, ist, dass sich die Anfragehäufigkeiten[8] der Daten ändern. Eines der Hauptkriterien, welches wir zur Verteilung unserer Daten im SAN heranziehen, ist die Häufigkeit mit der die Daten angefragt werden. Doch die Anfragehäufigkeit kann sich ändern, die Popularität der Daten ändert sich oder neue Daten lösen die bestehende ab. All dies sorgt dafür, dass unsere Daten nichtmehr optimal verteilt sind, und erfordert dass wir die Daten umsortieren[9]. Wir müssen unser Datenverteilung[10] an die neue Verteilung der Anfragehäufigkeit anpassen, um weiterhin alle Datenanfragen optimal bedienen zu können.

Da in der Zeit der Datenumschichtung[11] kein sicherer Speicherort unserer Daten vorhanden, ist steht das komplette Speichernetz solange nicht zur Verfügung bis unser System fertig angepasst ist. Um also die Effizienz unseres Speichernetzwerks zu erhalten muss dieser Überführungsprozess möglichst schnell abgeschlossen werden.

Der naive Ansatz ist hier, das bestehende Layout[12] in das neue Layout zu überführen. Dass heißt, die einzelnen Daten nacheinander so zu verschieben, bis das neue Layout erreicht ist.

Diese Überführung kann zum Problem werden, da die neue Verteilung komplett anders aufgebaut sein kann wie die bestehende, was den Überführungsprozess erschwert. Teilweise haben die verschiedenen Belegungen keinerlei Übereinstimmungen mehr, so dass wir die komplette Verteilung erneuern müssen. Da wir nun während der Überführung nicht mit dem Speichernetzwerk arbeiten können ist unser System in dieser Zeit nicht verfügbar.

One-Round-Migration

Unser Ziel ist es jetzt, nicht eine bestehende Verteilung in eine neue zu überführen, sondern nur Daten zu vertauschen welche die Netzwerkausnutzung erhöhen. Durch mehrfaches ausführen dieser Operation wollen wir unsere bestehende Datenverteilung möglichst gut an die neue Situation anpassen, und

[6] Optimale Ausnutzung des Speicherplatzes

[7] Optimierungsproblem aus dem Bereich der Kombinatorik - Aus einer Menge von Objekte sollen diese gewählt werden, welche unseren Nutzen maximieren, allerding das Gesamtgewicht nicht überschreiten

[8] entspricht der Anzahlt, wie oft eine Datei angefragt wird

[9] den neuen Anfragehäufigkeiten anpassen

[10] Verteilung der Daten auf den einzelnen Clientrechnern

[11] Verschiebung der Daten zu ihrem neuen Speicherort

[12] Muster der Datenverteilung auf unseren Clientsystemen

dabei deutlich weniger Zeit in Anspruch nehmen, das System sperren, wie bei dem naiven Ansatz.

Formale Definition

Bevor wir in die Details gehen müssen einige Punkte definiert werden:

Wir haben ein Speichernetzwerk mit N parallelen Clientsystemen[13]. Jeder Client i hat eine Kapazität K_i und eine Bandbreite L_i. Wir verteilen insgesamt m Daten, mit jeweils einer Anfragehäufigkeit l_i. Da wir die Dateigröße außer Acht lassen wird diese im Weiteren mit 1 angenommen, die Kapazität der Datenträger mit 3. Auch werden Daten immer komplett verteilt, wir nehmen also keine Segmentierung vor.

Aus unserem Anfragemuster[14], erhalten wir einen Plazierungsvektor $p_i = (p_{i1}, \ldots, p_{im})$, welche an der Stelle $p_{ij} = 1$ ist, wenn die Datei j auf i und ansonsten 0, und einen Anfragevektor $d_i = (d_{i1}, \ldots, d_{im})$ welcher uns den Anteil der Anfragen einer Datei j zurückgibt, welche auf Datenträger i gespeichert sind.

Um den benötigten Netzwerkflussgraphen aufzubauen definieren wir uns noch zwei Funktionen. Die Funktion $I(i)$ liefert uns alle Daten welche auf i gespeichert sind und $demand(j)$ gibt uns die aktuelle Anfragehäufigkeit der Datei j zurück.

Als Beschränkung für unseren Algorithmus müssen noch folgende Bedingungen eingeführt werden. Die Kapazität so wie die Bandbreite der auf i gespeicherten Daten dürfen nicht die Gesamtkapazität, so wie die Gesamtbandbreite, überschreiten.

Wir definieren uns eine Vertauschoperation als $\Delta(s_d, s_i, t_d, t_i)$, wobei s_d und t_d die Quell und Zieldatenträger und s_i und t_i die Quell und Zieldatei sind.

Da unser Speichernetzwerk während der Verschiebung nicht zur Verfügung steht, versuchen wir so viele Verschiebungen parallel auszuführen, wie möglich. Eine Gruppe solcher Verschiebungen, welche parallel ausgeführt werden, wird als Runde bezeichnet. Für eine Runde gilt, dass jeder Datenträger nur eine Operation, entweder Quelle oder Ziel einer Vertauschung, entgegennehmen kann.

Algorithmus

Unser erstes Ziel ist es, diejenige Verschiebung zu bestimmen, welche für uns den größten Nutzen erwirtschaften würde.

Dafür spannen wir einen Netzwerkflussgraphen (Abbildung 9) über alle Daten und Datenträger auf. Der Netzwerkflussgraph beginnt rechts mit dem Source-Knoten S, gefolgt von allen Daten a_j ($j \in [1, \ldots, m]$). Auf den Kanten (s, a_j) steht die neue Anfragehäufigkeit der Daten. Rechts daneben folgen alle Datenträger d_i ($i \in [1, \ldots, N]$). Es existieren Kanten (a_j, d_i), sofern die Datei in der aktuellen Belegung auf dem Datenträger liegt. Auf den Kanten von den Datenträgern zum Zielknoten T liegt das Gewicht der aktuellen Netzwerkausnutzung gemessen an der aktuellen Belegung und den neuen Dateianfragen.

Wir errechnen eine optimale Vertauschung in dem wir Phasenweise uns jeweils zwei Datenträger nehmen, d_x und d_y, und für jedes Tupel an Daten (a_x, a_y), aus $I(d_x) \times I(d_y)$, a_y auf d_y mit a_x überschreiben. Wir errechnen den nun entstehen max-flow[15] und speichern diesen mit der durchgeführten Vertauschung ab. Daten, welche nur einmal in Netzwerk vorhanden sind können nicht überschrieben werden, diese sind von diesen Operationen ausgeschlossen.

Haben wir alle Datenträgerpaarungen durch nehmen wir die Vertauschopertion, welche uns den größten max-flow liefert. Wir merken uns diese Vertauschung, und sperren diese Datenträgern, anschließend wiederholen wir den Suchvorgang auf den restlichen Datenträgern. Damit fahren wir solange fort bis wir entweder keine Datenträger mehr haben oder den max-flow nicht weiter erhöhen könne. Anschließen führen wie alle gespeicherten Operationen aus. Dies entspricht einer Runde unseres One-Round-Migration Algorithmus. Da wir jeden Datenträger sperren, sobald er in einer Operation vorkommt, wird kein Datenträger doppelt belastet. Somit können wir aller Verschiebungen in einer Runde abschließen.

Beispiel

Betrachten wir nun unseren Algorithmus an folgendem Beispiel aus [2]. Wir verteilen neun Daten (A,. . . ,I) auf drei Datenträger. Jede Datei hat eine Dateigröße von 1, die Dateigröße soll nicht weiter behandelt werden, und jeder Datenträger hat eine Kapazität von 3 und eine Bandbreite von 100.

Wir sehen in Abbildung 3 die aktuelle, und die neue, Anfragehäufigkeit.

Abbildung 4 zeigt uns die aktuelle Daten- und Nachfrageverteilung, Abbildung 5 die aktuelle Verteileilung mit dem neuen Anfragemuster.

Errechnen wir nun mittels des neuen Anfragemusters eine neue optimale Datenplatzierung würde dies wie in Abbildung 6 gezeigt aussehen.

[13] Rechner, welche die Daten speichern - ähnlich der Festplatten beim RAID-System

[14] Anfragenhäufigkeit unser Daten

[15] Maximaler Fluss unseres Netzwerkflussgraphen

Item	Initial demand	New Demand
A	130	55
B	90	55
C	40	20
D	30	60
E	25	5
F	25	10
G	25	15
H	22	70
I	13	110

Abbildung 3: Alte und neue Anfrageverteilung unserer Daten

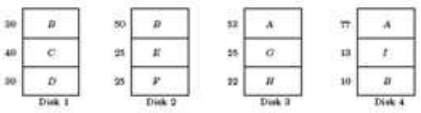


Abbildung 4: Datenverteilung mit der initianelen Anfrageverteilung

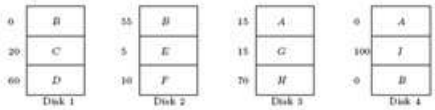


Abbildung 5: Altes Layout mit neuer Anfrageverteilung

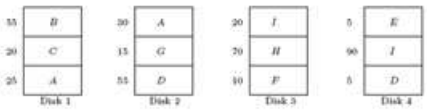


Abbildung 6: neue optimale Datenverteilung für das neue Anfragemuster

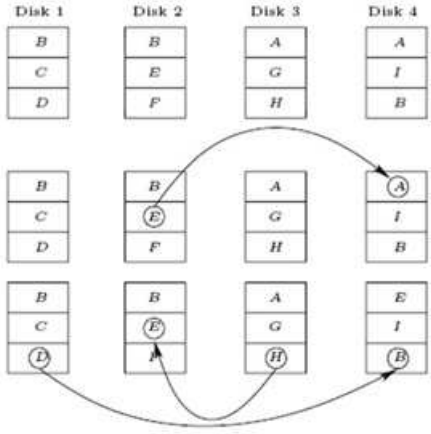


Abbildung 7: Schritt 1 und 2 für die Anpassung unseres Layouts an die neuen Anfragen

Um unsere bestehende Datenverteilung in das neue Layout aus Abbildung 6 zu überführen, sind die vier Schritte aus Abbildung 7 und 8 notwendig.

Alternativ können wir mittels unseres One-Round-Migration Verfahrens und des Netwerkflussgraphen (Abbildung 9) eine optimale Verschiebung errechnen (Abbildung 10), welche unser bestehendes Layout an die neue Anfageverteilung anpasst.

Wir sehen nun, dass wir mit einer Verschiebung unser bestehendes Layout an die neue Anfragehäufigkeit anpassen können, und dabei mit nur einer Runde ein identisches Ergebnis erziehlen wie mit einer Neuberechnung unserer Verteilung und anschließender Überführung.

Speed-Up

Wie wir im Beispiel gesehen haben, können wir so ein relativ identisches Ziel mit deutlich geringeren Verschiebeoperationen erreichen. Wir reduzieren die, meist recht Zeitaufwendigen Datentransfers und führen die verbleibenden parallel aus, so dass wir den gesamten Zeitaufwand der Anpassung an das neue Anfragemuster verringern. Allerdings erhöhen wir auch den Rechenaufwand. Wir müssen nun für jede Runde eine ganze Menge rechenintensiver Schritte durchlaufen. Um unser Verfahren auch bei großen Datenmengen effizient zu halten, müssen wir versuchen den Rechenaufwand zu reduzieren.

Speicherung des Netzwerkflussgraphen

Ein Punkt, welcher und großen Rechenaufwand bereitet, ist das wir bei jeder Vertauschung den kompletten Netzwerkflussgraphen berechnen müssen. Allerdings ändert sich immer nur ein kleiner Teil beim durchführen einer Verschiebung. Daher erreichen wir eine deutliche Reduktion der Berechnungen, wenn wir die flow-Values[16] unseres Netzwerkflussgraphen speichern, und unsere Verschiebeoperationen darauf zu berechnen.

Reduzierung der Berechnungen

Eine weitere Variante ist die Reduktion, der zur Berechnung herangezogenen Daten. In der Originalvariante haben wir unsere Tupel aus allen Daten, welche auf den beiden Datenträgern gespeichert sind, gebildet.

Variante 1

Wir können den Rechenaufwand verringern indem wir uns ein $I_+(d_i)$ definieren, welches alle Datenträ-

[16] Flusswerte unseres Netzwerkflussgraphen

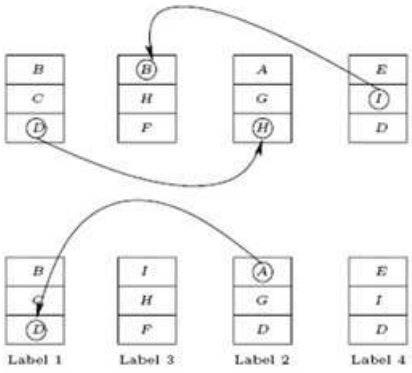


Abbildung 8: Schritt 3 und 4 für die Anpassung unseres Layouts an die neuen Anfragen

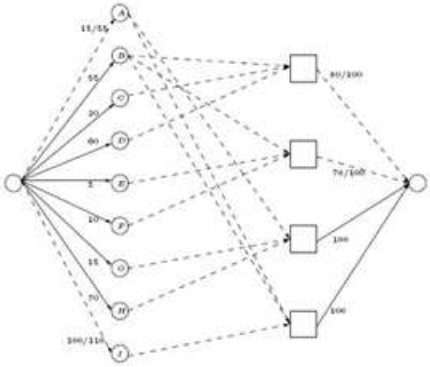


Abbildung 9: Netzwerkflussgraph über unser Daten und Datenträger

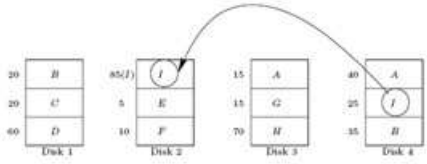


Abbildung 10: Optimale Verschibung

ger beinhaltet, deren Anfrage zum aktuellen Zeitpunkt nicht, oder nicht komplett, gedeckt wird. Zur Verschiebung ziehen wir nun nurnoch Tupel (a_x, a_y) aus $I_+(d_x) \times I(d_y)$ heran.

Variante 2

Eine weitere Variante ist, das wir auf dem Zweiten Datenträger nur Daten heranziehen, die eine sehr geringe Anfrage haben. Wir bekommen diese über $I_-(d_i)$, und bilden somit Tupel (a_x, a_y) aus $I_+(d_x) \times I_-(d_y)$.

Experimente

Wir stellen uns nun die Frage, was bringt uns dieses Verfahren unter realistischen Bedingungen?

Dafür wurden einige Experimente durchgeführt. Der Ablauf der Experimente sieht wie folgt aus. Wir generieren durch einen Zufallsgenerator eine Anfrageverteilung unseren Daten. Anschließend ermitteln wir eine Verteilung der Daten mittels des Sliding-Window-Algorithmus. Wieder per Zufallsgenerator generieren wir unser neues Anfragemuster.

Bei der ersten Messreihe bestimmen wir die Anzahl der benötigten Runden um unsere besehende Verteilung in die neue zu überführen. Anschließend lassen wir unseren One-Round-Migration Algorithmus 10 Runden laufen, und bewerten den prozentualen Anteil welchen wir erreichen konnten, gemessen am Optimum.

Wir führen insgesamt 4 verschieden Veränderungen der Anfragehäufigkeit durch. Bei der ersten Variante wird die original Anfragehäufigkeit linear steigend verteilt, und anschließend steigern wir von 20% unserer Daten die Popularität, welche daraufhin anders gelagert werden müssen. Variante 2 startet ebenfalls mit einer linear steigenden Verteilung, dreht diese aber für die neue Verteilung um. Folglich werden Daten, welche vorher selten angefragt worden später stärker angefragt, stark angefragte Daten äquivalent weniger. Variante 3 beginnt damit, dass jede Datei gleich oft angefragt wird, und wandelt diese in eine linear ansteigende Verteilung um. Die Gegenoperation dazu, von einer linear steigenden Anfragehäufigkeit in eine identische Anfragehäufigkeit wird in Variante 4 behandelt.

Ausgeführt werden diese Experimente jeweils auf 2 Speichersystemen. Beim ersten System haben wir immer 60 Datenträger, einmal mit einer Kapazität von 15 und einer Bandbreite von 40, einer Kapazität von 30 und einer Bandbreite von 35 und einmal mit einer Kapazität von 60 und einer Bandbreite von 150. Bei unserem zweiten System haben wir immer Datenträger der Kapazität 60 und der Bandbreite

150, allerding erhöhen wir die Anzahl der Datenträger in 10er Schritten von 10 bis auf 100.

Abbildung 11 zeigt uns die benötigten Runden bis die Algorithmen terminieren. Unser One-Round-Migration wird jeweils immer nur 10 Runden ausgeführt. Die Abblildungen 12, 13 und 14 zeigen uns die prozentualen Werte, welche mit der One-Round-Migration erreicht werden, so wie die Vergleichwerte zur naiven Methode.

Bei der zweiten Versuchsreihe betrachen wir unseren One-Round-Migration Algorithmus auf dem Datenträgern, welche in der Anzahl jeweisl um 10 erhöht werden. In den Abbildungen 15, 16, 17 und 18 sind die Messergebnisse aufgeführt. Besonderst auffällig ist hier das Verhalten unseres Algorithmus, so wie der naiven Methode, bei steigender Datenträgeranzahl.

Die Ergebnisse stammen aus dem Paper [2].

Literatur

[1] "How much information?" - School of Information Management and Systems. University of California at Berkeley. - http://www.sims.berkeley.edu/research/projects/howmuch-info 2003/

[2] S. Kashyap, Y. Wan and L. Golubchik. Fast reconfiguration of data placement in parallel disks. In Proceedings of the 7th Workshop on Algorithm Engineering & Experiments (ALENEX'06), 2006

[3] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, pages 223-232, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

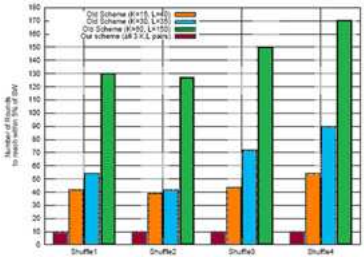
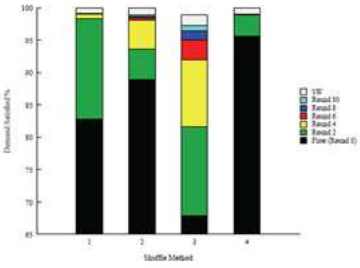
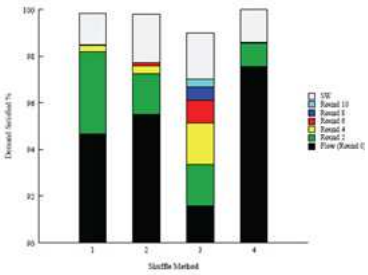


Abbildung 11: Anzahl an benötigten Runden. 1. Speichersystem und die vier Veränderungen der Anfragehäufigkeit



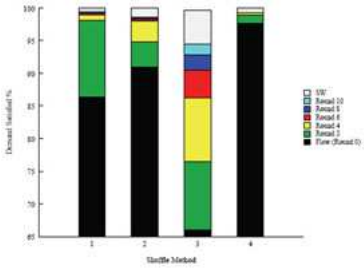
Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	10	99.10	41.8	99.92
2	10	98.86	59.1	99.92
3	10	97.26	43.7	98.91
4	10	99.04	54.2	100

Abbildung 12: One-Round-Migration auf Datenträger K=15; L=40



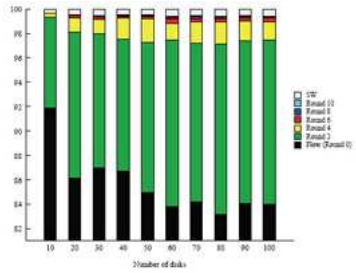
Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	10	98.50	54.2	99.83
2	10	97.72	41.6	99.79
3	10	97.02	71.8	98.99
4	10	98.57	89.9	100

Abbildung 13: One-Round-Migration auf Datenträger K=30; L=35



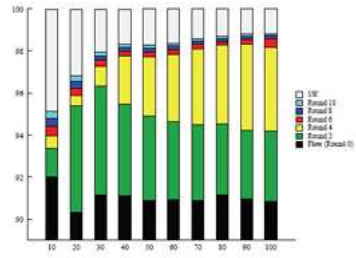
Shuffle method	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
1	10	99.41	130.3	99.98
2	10	98.54	127.3	99.99
3	10	94.41	150.4	99.68
4	10	99.30	170.5	100

Abbildung 14: One-Round-Migration auf Datenträger K=60; L=150



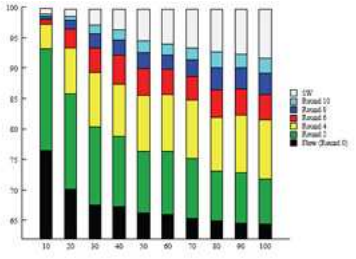
N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	99.64	**104.8**	99.98
20	**10**	99.52	**111.8**	99.99
30	**10**	99.50	**121**	99.99
40	**10**	99.53	**121.9**	99.98
50	**10**	99.51	**125.9**	99.99
60	**10**	99.41	**128.2**	99.98
70	**10**	99.46	**128.5**	99.99
80	**10**	99.42	**135.4**	100
90	**10**	99.45	**138.6**	99.99
100	**10**	99.43	**136.2**	99.99

Abbildung 15: One-Round-Migration auf Datenträger 1 - Datenverteilung 1. Fast am Optimum, trotz steigender Datenträgeranzahl



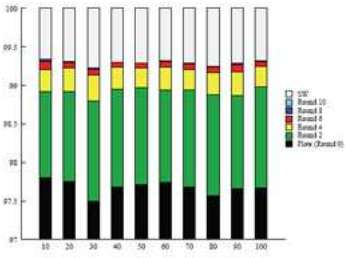
N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	95.12	**103.3**	99.98
20	**10**	96.82	**109.4**	99.98
30	**10**	97.96	**119.4**	99.97
40	**10**	98.31	**119.7**	99.98
50	**10**	98.26	**124.7**	99.99
60	**10**	98.37	**127.4**	100
70	**10**	98.57	**129.3**	100
80	**10**	98.67	**135.3**	100
90	**10**	98.81	**134.4**	100
100	**10**	98.82	**137.5**	100

Abbildung 16: One-Round-Migration auf Datenträger 1 - Datenverteilung 2. Effizienz steigt mit steigender Anzahl an Datenträgern



N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	98.89	**126.2**	99.75
20	**10**	98.46	**133**	99.71
30	**10**	97.03	**138.8**	99.7
40	**10**	96.22	**144.1**	99.68
50	**10**	94.45	**146.5**	99.69
60	**10**	93.80	**149.4**	99.67
70	**10**	93.29	**149.8**	99.68
80	**10**	92.69	**154.6**	99.66
90	**10**	92.29	**152.8**	99.65
100	**10**	91.63	**155.9**	99.66

Abbildung 17: One-Round-Migration auf Datenträger 1 - Datenverteilung 3. Sinkende Effizienz mit Steigender Anzahl an Datenträgern



N	Our Scheme		Old Scheme	
	Rounds	Demand %	Rounds (avg)	Demand %
10	**10**	99.33	**128.3**	100
20	**10**	99.31	**139.6**	100
30	**10**	99.22	**148.9**	100
40	**10**	99.30	**159.2**	100
50	**10**	99.29	**166.6**	100
60	**10**	99.32	**170.8**	100
70	**10**	99.29	**178.7**	100
80	**10**	99.25	**183.6**	100
90	**10**	99.29	**190.3**	100
100	**10**	99.32	**196.1**	100

Abbildung 18: One-Round-Migration auf Datenträger 1 - Datenverteilung 4. Effizienz trotz steigeder Datenträgeranzahl konstant