# ClusterRAID: Architektur und Prototyp eines verteilten fehlertoleranten Massenspeichersystems für Cluster

Arne Wiebalck

Lehrstuhl für Technische Informatik
Universität Heidelberg
arne.wiebalck@web.de

**Abstract:** Diser Artikel beschreibt die Architektur und die Prototyp-Implementierung eines verteilten, fehlertoleranten Massenspeichersystems für Cluster. Die grundlegende Idee der Architektur ist es, die lokale Festplatte eines Clusterknotens zuverlässig zu machen, ohne dabei die Schnittstelle für das Betriebssystem bzw. für die Anwendung zu verändern. Hierbei werden fehler-korrigierende Codes eingesetzt, die es ermöglichen, die Anzahl der zu tolerierenden Fehler und somit die Zuverlässigkeit des Gesamtsystems beliebig einzustellen. Das Anordnungsschema für die Datenblöcke innerhalb des Systems berücksichtigt das Zugriffsverhalten einer ganzen Klasse von Applikationen und kann so die erforderlichen Netzwerkzugriffe auf ein Minimum reduzieren. Gründliche Messungen und Funktionstests des Prototypen, sowohl allein als auch im Zusammenwirken mit lokalen und verteilten Dateisystemen, belegen die Validität des Konzeptes.

## 1 Cluster als Supercomputer

Cluster aus Standard-Komponenten haben sich in den letzten Jahren mehr und mehr als die dominante Architektur für Supercomputer-Installationen durchgesetzt. Dies belegt etwa die aktuelle TOP500-Liste, das Ranking der 500 leistungstärksten Rechner weltweit [TOP]: Fast drei Viertel der aufgeführten Systeme sind cluster-basiert. Hauptgründe für die zunehmende Verbreitung dieser Architektur seit den frühen 1990er Jahren sind das ausgesprochen gute Preis-Leistungsverhältnis sowie die gesteigerte Verfügbarkeit und die enorme Leistungsteigerung von Hard- und Software. Insbesondere sind hier Netzwerk-Technologien und Mikroprozessoren auf der einen, Betriebsysteme und Programmier-Werkzeuge auf der anderen Seite zu nennen.

Wegen dieser Stärken werden Cluster in Wissenschaft und Industrie gleichermassen genutzt. So sind Cluster in der Größenordnung von 1000 PCs als Teil der Auslesekette und zur Auswertung der Daten für die geplanten Experimente der Hochenergie-Physik des Large Hadron Colliders [LHC] am Europäischen Kernforschungszentrum CERN[CER] vorgesehen. Auch die wohl beliebteste Web-Suchmaschine Google [Goo] verwendet Cluster, um ihre rund 8 Milliarden Webseiten zu indezieren und Anfragen zu beantworten. Insgesamt umafsst sie von Google eingesetzte Cluster-Architektur mehr als 15000 aus Standard-Komponenten aufgebaute PCs [BDH03].

Aufgrund eines zu beobachtenden Paradigmen-Wechsels von rein rechenintensiven hin zu Eingabe-/Ausgabe-intensiven Anwendungen werden die in Clustern eingesetzten Massenspeichersysteme zu einer immer wichtigeren Komponente. Daß sich bisher kein Standard für die Anbindung oder die Nutzung von Massenspeicher in Clustern herausbilden konnte, liegt zum einen an der inhärenten Unzuverlässigkeit dieser Architektur – insbesondere bei den sogenannten *commodity off-the-shelf* Clustern, also aus Standard-Bausteinen aufgebauten Systemen –, zum anderen an dem breiten Anforderungsspektrum, das ein ebenso breites Spektrum an Lösungen hervorgebracht hat.

Viele Massenspeichersysteme sind individuell angepasste Lösungen: So hat Google zum Beispiel sein eigenes Cluster-Dateisystem GoogleFS [GGL03] entwickelt, das der erwähnten Unzuverlässigkeit der Hardware durch Replizierungs- sowie Fehlerkorrekturmechanismen Rechnung trägt. Die speziellen Anforderungen bei Google fanden im Design von GoogleFS ebenso Berücksichtigung: die typische Dateigröße bei Google von mehreren Gigabytes beeinflusste Dateisystem-Parameter wie die verwendete Blöckgröße oder die Implementierung von E/A-Operationen. Auch das Zugriffsmuster der Anwendung ging in das Design ein. Die Dateien bei Google werden in der Regel nur einmal geschrieben (und nur selten verändert), um dann mehrfach gelesen zu werden, typischerweise sequentiell. Für solch ein Zugriffsverhalten ist etwa das Caching auf der Client-Seite der Performance eher abträglich. Da Suchanfragen bei Google völlig unabhängig voneinander behandelt werden können, ergibt sich ein hohes Maß an intrinsischer Parallelität, welches Google durch parallel arbeitende Index-Server, die unabhängige Datensätze bearbeiten, ausnutzt. Jede Anfrage löst typischerweise das Durchsuchen von mehreren hundert Megabytes an Daten aus. Die Zahl der Lesezugriffe überwiegt die Zahl der Schreibzugriffe also bei weitem. Die dreifach redundante Speicherung der Daten erlaubt nicht nur einen Schutz gegen Datenverlust, sondern ermöglicht auch eine Lastverteilung.

Da die Daten der CERN-Experimente in voneinander unabhängigen sogenannten Ereignissen anfallen, ist hier ebenfalls eine massiv parallele Verarbeitung möglich. Technisch realisiert wird dies durch eine hierarchisches System von verteilten Analyse-Zentren, dem Tier-Modell [BCH+01]. Kopien der Daten werden in mehreren Stufen weltweit auf Rechenzentren verteilt, um dort den Wissenschaftlern zur Verfügung zu stehen. Neben der Möglichkeit der parallelen Verarbeitung und der unbedingten Notwendigkeit, sich wegen der immensen Kosten der Experimente vor einem eventuellen Verlust der Daten zu schützen, ähnelt diese Anwendung auch in der Bevorzugung der Lese- vor den Schreibzugriffen der Suchmaschine Google, denn die Daten der Experimente werden immer mehrfach verarbeitet und das Ergebnis einer Analyse ist in der Regel deutlich kleiner als die Eingangsdaten.

Cluster-Anwendungen mit einem Speicherbedarf im Petabyte-Bereich und Installationen mit mehreren Tausend Knoten sind heute erst im Entstehen. Die Ähnlichkeiten der beiden obigen Beispiele, nämlich die Forderung nach fehlertoleranter Speicherung in unzuverlässigen Systemen, die intrinsische Parallelität der einzelnen Aufgaben, die Notwendigkeit eines optimalen Preis-Leistungsverhältnisses oder die Neigung zu mehr Lese- als Schreibzugriffen werden jedoch von vielen Anwendungen geteilt. Vorhandene Massenspeichersysteme erfüllen diese Anforderungen nur zum Teil, und deshalb versucht die im folgenden vorgestellte Architektur [Wie05, LW03] diese Lücke zu schliessen.

## 2 ClusterRAID Architektur

Bei der Entwicklung der ClusterRAID-Architektur und beim Design der Prototyp-Implementierung fanden folgende Punkte Berücksichtigung:

- Optimierung für den Lesezugriff,
- Ausrichtung auf Applikationen, deren Datenzugriff parallelisierbar ist,
- einstellbare Fehlertoleranz,
- eine Schnittstelle auf Block-Ebene,
- Reduzierung des benötigten Plattenplatzes,
- Reduzierung des Netzwerkverkehrs,
- Reduzierung des anfallenden Prozessor-Overheads,
- Skalierbarkeit für größere Systeme, und
- der Einsatz unter GNU/Linux auf Intel-basierten Prozessoren.

Die ersten beiden Punkte wurden bereits im vorigen Kapitel motiviert, ebenso wie die Notwendigkeit eines Konzeptes zur Fehlerbehandlung. Die Einstellbarkeit der Fehlertoleranz, d.h. die Möglichkeit, die Wahrscheinlichkeit für den Datenverlust über System-Parameter kontrollierbar zu machen, beeinflusst auch den Punkt der Skalierbarkeit, da die Wahrscheinlichkeit für Fehler mit der Zahl der in einem System verwendeten Komponenten zunimmt. Auch der Platzbedarf und seine Minimierung sind von diesem Punkt betroffen, da eine höhere Datensicherheit unbedingt mit einem größeren Platzbedarf zur Speicherung von Redundanzinformationen einhergeht. In der Theorie fehlerkorrigierender Codes wird dieser Sachverhalt durch die Singleton-Grenze ausgedrückt [HHL+00]. Nicht zuletzt hängt auch die benötigte Rechenleistung und der Netzwerkverkehr von den verwendeten Mechanismen zum Schutz vor Datenverlust ab, weil Codes mit höherem Schutz in der Regel aufwendiger zu berechnen sind und auch mehr Redundanz-Informationen erzeugen, die in verteilten Systemen zu übertragen sind.

Die grundlegende Idee der ClusterRAID-Architektur zur Umsetzung dieser Anforderungen ist es, durch geeignete Mechanismen den *lokalen* Massenspeicher eines Clusterknotens hinreichend zuverlässig zu machen, ohne dabei die (blockartige) System-Schnittstelle des Speichers zu verändern. Ein ClusterRAID Device ist ein virtuelles Gerät, eine Abstraktionsschicht, die oberhalb des eigentlichen, physikalischen Gerätes liegt. Im Normalbetrieb werden alle Speicheranfragen an das zugeordnete physikalische Specihermedium weitergereicht. Aus Sicht der Anwendung verhält sich ein ClusterRAID Device genau wie die lokale Festplatte. Tatsächlich werden jedoch alle Zugriffe unterbrochen. Im Falle von Lesezugriffen wird die Anfrage an den zugeordneten Speicher weitergegeben und auf ihrem Rückweg auf Fehler überprüft. Bei Schreibzugriffen erzeugt das ClusterRAID Device zusätzlich Redundanz-Informationen, die im Cluster verteilt werden. Im Falle von

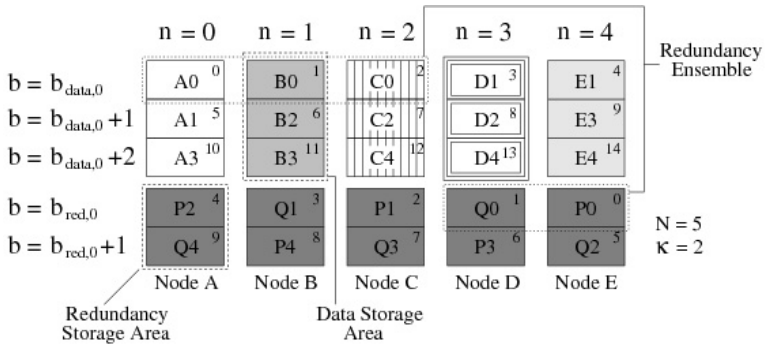


Abbildung 1: Datenverteilung bei verteilter Redundanz. Alle Festplatten (oder Knoten) A bis E speichern sowohl Daten- als auch Redundanzblöcke. Die Zahlen kennzeichnen ihre Nummerierung ($\lambda_{data}$,$\lambda_{red}$), die für die Abbildung von einem Datenblock auf seine zugehörigen Redundanzblöcke genutzt wird. Die Blocknummern ($b_{data,0}$ and $b_{red,0}$ kennzeichnen den jeweiligen Anfang eines Speicherbereiches.

Fehlern beim Datenzugriff können die zur Zeit nicht verfügbaren Daten dann aus den verteilten Redundanzinformationen rekonstruiert werden. Dieser Ansatz ähnelt demjenigen von Festplatten-Arrays, den *Redudant Arrays of Inexpensive Disks* (RAID) [PGK88], und stellt in gewissem Sinne eine Erweiterung dieses Konzeptes für den Schutz von Daten auf die Cluster-Ebene dar. Abgesehen davon, dass es sich beim ClusterRAID um ein verteiltes System handelt, unterscheidet es sich in der Daten- oder Blockverwaltung von den klassischen RAIDs. Logisch zusammenhängende Daten werden nicht über mehrere Devices verteilt (Striping), sondern genauso auf dem Speicher gesichert, als sei die lokale Festplatte das einzige Speichermedium im Cluster. Diese Art der Datenverteilung hat den Vorteil, daß lesende Zugriffe ohne jeden Netzwerkverkehr auskommen. Weiterhin kann selbst in dem Fall, dass mehr Speicher ausfallen als die eingesetzten fehlerkorrigierenden Codes tolerieren können, auf die Daten auf den verbliebenen Festplatten problemlos zugegriffen werden.

Die Gesamtmenge aller im Cluster vorhandenen Speicherblöcke wird beim ClusterRAID in Speicherbereiche für Daten und Speicherbereiche für Redundanzinformationen geteilt. Ein Datenspeicherbereich liegt dabei immer vollständig auf einem Knoten und steht ihm zur exklusiven Nutzung zur Verfügung. Jeder Block gehört zu einem Redundanz-Ensemble, welches neben den Datenblöcken auch die zugehörigen Redundanzblöcke einschliesst. Die Verteilung der Redundanzblöcke im Cluster ist beliebig, mit der Einschränkung, dass nie mehr als ein Block eines Ensembles auf demselben Speicher abgelegt ist. Ein Speicherausfall würde dann nämlich in einen mehrfachen Fehler auf Blockebene zur Folge haben und den Schutz vor Datenverlust erheblich mindern. Die Abbildung 1 zeigt als Beispiel eine mögliche Verteilung von Daten- und Speicherbereichen für einen Cluster mit $N = 5$ Knoten und $\kappa = 2$ Redundanzblöcken pro Ensemble.

Abbildung 2 zeigt einen funktionalen Überlick der ClusterRAID-Architektur. Die Anwendung greift über Schreib-Lese-Module auf das ClusterRAID zu. Diese Module liefern ein

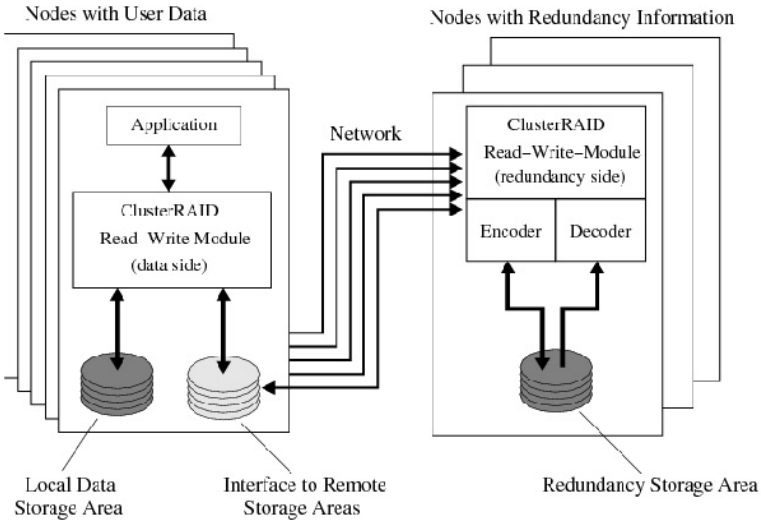


Abbildung 2: Funktionaler Überblick über die ClusterRAID Architektur.

exaktes Abbild des darunterliegenden physikalischen Devices. Bei Schreibzugriffen nutzt das ClusterRAID die Technik des sogenannten Netzwerkblock Devices (NBD) [BLA00] um die Daten zu einem entfernten Speicherbereich zu transferieren. Teil des Datenpfades ist außerdem ein Codec, der einen fehlerkorrigierende Code auf die Daten anwendet. Die Lage des Codecs im Datenpfad – also auf Seite des Datenspeicherbereiches oder auf der Redundanzseite – ist dabei beliebig. Derselbe Mechanismus des Datentransfers wird auch verwendet, um im Fehlerfall die Daten aus den Redundanzspeicherbereichen in den Knoten mit dem fehlerhaften Speicher zu bringen, um dort die angeforderten Daten zu rekonstruieren.

## 3 Implementierung, Leistungs-Messung und funktionale Tests

Der Prototyp des ClusterRAIDs wurde durch einen Satz von Kernel-Modulen für das GNU/Linux Betriebssystem realisiert. Die Implementierung umfasst dabei drei Module. Das Haupt-Modul bildet die Schnittstelle zum Kernel, bedient die Anfragen der Anwendungen und steuert das Thread-Handling. Das Kodierer-Modul implementiert Vandermonde-basierte Reed-Solomon Codes [Pla97, PD05] auf Kernel-Ebene und das XOR-Modul nutzt die SIMD-Befehle des Prozessors aus, um eine Beschleunigung der vielfach verwendeten XOR-Operation beim Zugriff auf das ClusterRAID zu erreichen. Zusätzlich umfasst das ClusterRAID eine XML- und Ruby-basierte Kontroll-Software.

Abbildung 3 zeigt den Netzwerkverkehr und den ClusterRAID-Durchsatz während eines simulierten Festplatten-Ausfalls. Zu Beginn des Tests greift die Applikation lesend auf

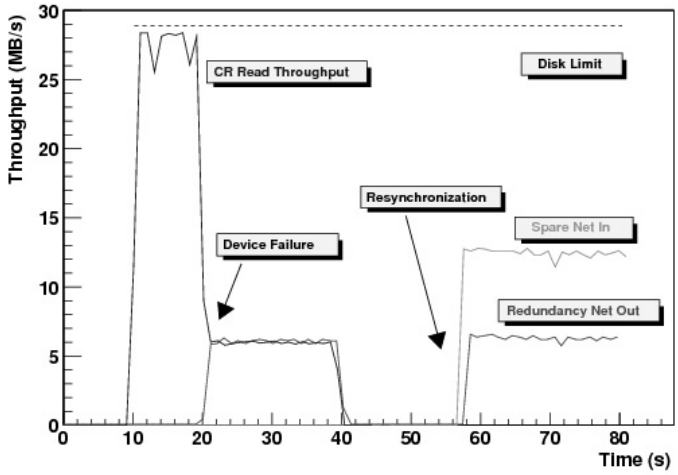


Abbildung 3: Netzwerk-Trace zur Veranschaulichung der Online-Rekonstruktion des ClusterRAIDs bei einem Festplattenausfall.

das ClusterRAID Device zu. Der Durchsatz zu diesem Zeit beträgt wegen des sehr geringen zusätzlichen Overheads der ClusterRAID-Schicht im Betriebssystem-Kernel über 95% des nominellen Festplattendurchsatzes. Etwa 20 Sekunden nach Beginn der Aufzeichnung wird dem ClusterRAID der Zugriff auf die lokale Festplatte verweigert. Die Applikation wird vom ClusterRAID allerdings weiter mit den angeforderten Daten versorgt, die das System nun aus den verteilt abgelegten Redundanzinformationen rekonstruiert. Der ausgehende Netzwerkverkehr auf einem der Redundanz-Rechner bestätigt dies. Der Rückgang des Durchsatzes erklärt sich mit der Auslastung der CPUs durch die rechenintensive Rekonstruktion der Daten. Nach rund 40 Sekunden stoppt die Applikation den Zugriff. Je nach verwendetem Redundanz-Algorithmus bzw. seiner Konfiguration kann das ClusterRAID auch mehrere gleichzeitige Fehler ohne Datenverlust oder Unterbrechung der Anwendung kompensieren.

Das ClusterRAID ist in der Lage einen defekten Knoten durch einen neuen Knoten zu erstzen und die Daten des ausgefallenen Knoten auf dem Ersatz-Device zu rekonstruieren. Dieser Vorgang ist in der rechten Hälfte der Abbildung 3 zu sehen. Der Ersatzknoten erstellt Verbindungen zu den übrigen Knoten im ClusterRAID-Verbund und übernimmt die Funktion des ausgefallenen Knotens. Er rekonstruiert dann die Daten mittels der Redundanzinformation und der Daten der verbliebenen Rechner.

Der Aggregat-Durchsatz beim Lesen vom ClusterRAID skaliert perfekt mit der Zahl der beteiligten Knoten. Da sich das System wegen des lokalen Zugriffs in diesem Fall wie eine Menge völlig unabhängiger Knoten verhält, ist das nicht überraschend. Um beim Schreiben und der Berechnung der Redundanzinformation nicht auf die übrigen Daten eines Ensembles zugreifen zu müssen, verwendet das ClusterRAID die Differenz zwischen den

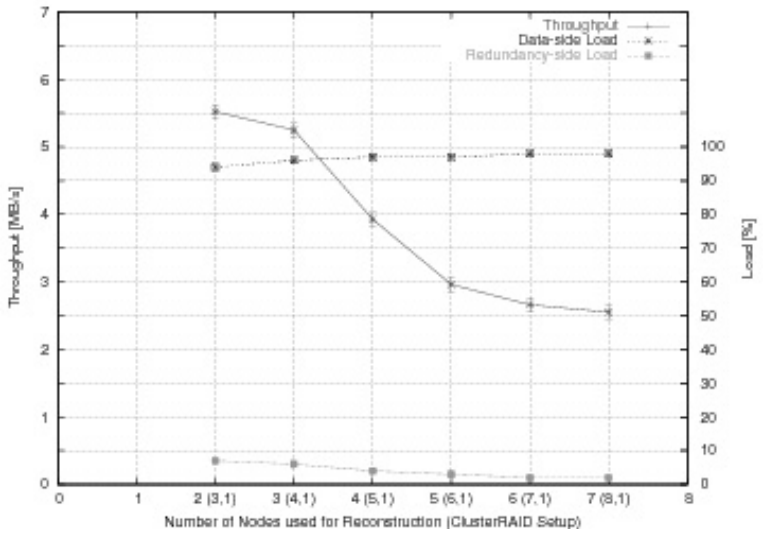


Abbildung 4: Durchsatz und Last in Abhänigkeit von der Zahl der bei der Rekonstruktion beteiligten Knoten.

bereits gespeicherten und den neu abzulegenden Daten eines Blockes, um die Änderung der Redundanzinformation zu bestimmen. Mit dieser Änderung werden die Redundanzblöcke eines Ensembles korrgiert. Nachteil dieses Ansatzes ist, dass jedem Schreibzugriff ein Lesezugriff vorausgeht, sowohl auf der Daten- als auch auf der Redundanzseite. Dies limitiert den Durchsatz auf maximal die Hälfte der lokalen Festplattenbandbreite. Außerdem erfordert die Nutzung der Differenzbildung, daß jeder Schreibzugriff tatsächlich auf den Redundanzblöcken reflektiert ist. Der Prototyp schreibt deshalb die Daten redundanzseitig im synchronen Modus, um unerwünschte Effekte durch den Block-Cache zu unterdrücken. Diese Synchronizität beeinträchtigt den Schreibdurchsatz erheblich, während das Abschalten der Locks im ClusterRAID oder die Verwendung des Mehrprozessorbetriebes hier kaum einen Einfluss hat. Insgesamt liegt der Durchsatz eines einzelnen schreibenden Knotens bei rund 6 MBytes/s.

Abbildung 4 zeigt Durchsatz und Last während der Rekonstruktion als Funktion der zur Rekonstruktion erforderlichen Knoten. Wie oben erwähnt limitiert die Leistung der CPUs hier eindeutig den Durchsatz. Die Last auf dem rekonstruierenden Knoten liegt immer jenseits von 90%. Die Rate, mit der die Anwendung auf die rekonstruierten Daten zugreifen kann, sinkt mit steigender Knotenzahl aus zwei Gründen. Zum einen erhöht sich bei steigender Knotenzahl die Prozessorlast durch den Netzwerkverkehr und reduziert die zur Berechnung der Reed-Solomon Codes zur Verfügung stehenden CPU-Zyklen, zum anderen erhöht sich die Rechenkomplexität mit der Zahl der Rechner im Ensemble.

Neben den Leistungs- und Funktionstest hat die Prototyp-Implementierung auch Stabilitäts- und Konsistenztests durchlaufen. So wurde etwa auf das reine Blockdevice über 10 Tage permanent lesend und schreibend zugegriffen oder mittels des E/A-Benchmarks

bonnie++ 10 Tage lang seine Leistung vermessen. Da die Rekonstruktion unbedingte Konsistenz der Daten voraussetzt, wurde die Korrektheit der Daten nach konkurrierenden Schreibzugriffen auf denselben Block, nach gleichzeitigen Schreibzugriffen oder nach mehrtägigem permanentem Zugriff überprüft. Auch die Überprüfung der Datenkonsistenz nach dem Ersatz eines ausgefallenen Rechners oder die Qualität der Daten während der Rekonstruktion – auch bei gleichzeitigem Ausfall mehrerer Festplatten – hat der Prototyp erfolgreich bestanden.

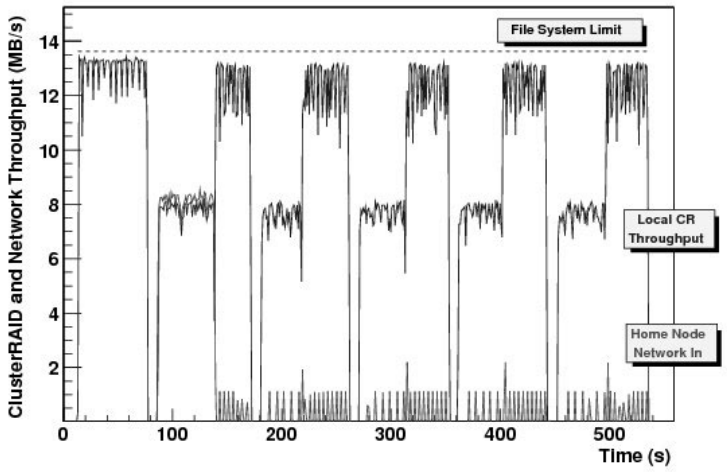


Abbildung 5: Zusammenspiel vom ClusterRAID mit MOSIX. Eine detaillierte Beschreibung findet sich im Text.

## 4 ClusterRAID und Cluster-Dateisysteme

Auf einem Knoten liefert das ClusterRAID keine globale Sicht auf die gesamten Daten im Cluster. Es bietet vielmehr ein beliebig zuverlässiges Block-Device für die lokale Speicherung von Daten an. Die Realisierung einer globalen Sicht auf die Daten wird den Cluster-Dateisystemen überlassen. Diese können auf das ClusterRAID als unterliegendes, zuverlässiges Block-Device aufsetzen, also selbst auf Mechanismen zur sicheren Speicherung der Daten verzichten. Diese ist auf Datei-Ebene ohnehin deutlich schwieriger effizient zu gestalten als auf Block-Ebene.

Um die spezielle Architektur des ClusterRAIDs, und hier insbesondere die Optimierung für Lesezugriffe, auszunutzen, müssen zudem Vorkehrungen getroffen werden, daß die Anwendung nach Möglichkeit nur auf lokal gespeicherte Daten zugreift. Hierfür ist also ein intelligenter Batch- oder Scheduler-Mechanismus nötig, der die Anwendung auf dem Knoten mit den von ihr benötigten Daten startet.

Um das Zusammenspiel des ClusterRAIDs mit einem Cluster-Dateisystem und einem solchen Scheduler zu untersuchen, wurde MOSIX [BGW93] mit seinem Cluster-Dateisystem MFS gewählt. MOSIX ist eine Betriebssystem-Erweiterung des Linux-Kernels für die automatische Prozess-Migration zum Zwecke des Lastausgleichs. Unter anderem geht in die Entscheidung, einen Prozess zu migrieren, auch sein E/A-Verhalten ein. Greift ein Prozess etwa wiederholt auf Daten in einem entfernten Knoten zu, so migiriert MOSIX diesen Prozess um das Versenden der Daten über das Netzwerk zu reduzieren und den Durchsatz auf die Daten für die Applikation zu erhöhen. MOSIX folgt also hier dem Paradigma, die Prozesse zu ihren Daten zu schicken und nicht umgekehrt. Da das ClusterRAID lokale Zugriffe bevorzugt, bietet MOSIX mit seiner globalen Datensicht und der Möglichkeit, alle Zugriffe durch Migration in lokale Zugriffe zu verwandeln, hier eine ideale Ergänzung.

Abbildung 5 zeigt Netzwerk-Verkehr und den ClusterRAID-Durchsatz aus Sicht einer Applikation in einem MOSIX-System mit einem ClusterRAID als Massenspeicher. Die Anwendung ist so entworfen, daß sie reihum auf die Daten verschiedener Rechner zugreift. Sie beginnt auf ihrem sogenannten Heimatknoten – das ist der Knoten, auf dem die Anwendung ursprünglich gestartet wurde, – und liest die Daten über das ClusterRAID annähernd mit dem Durchsatz des Dateisystems ein. Nach rund 80 Sekunden greift sie über das MFS auf einen entfernten Knoten zu. Die Daten werden über das Netzwerk in den Heimatknoten transferiert. Der Durchsatz ist im Vergleich zum lokalen Zugriff jedoch erheblich reduziert. Mit einer (einstellbaren) Latenz, migriert MOSIX den Prozess auf den Knoten, der die Daten vorhält. Nun kann der Prozess wieder lokal auf ein ClusterRAID Device zugreifen. Der Durchsatz ist deutlich erhöht und der Netzwerkverkehr auf nahezu Null zurückgegangen. Beim Zugriff auf einen dritten Knoten wiederholt sich der Vorgang. Neben der Verwendbarkeit des ClusterRAIDs mit einem Cluster-Dateisystem, zeigt dieses Beispiel sehr deutlich die Vorzüge einer Prozess-Migration und des lokalen Zugriffs bei daten-intensiven Anwendungen. Auch der Fall einer ausgefallenen Festplatte konnte erfolgreich mit MOSIX und dem ClusterRAID getestet werden.

## 5 Zusammenfassung

Das ClusterRAID ist eine verteilte Massenspeicher-Architektur, die die lokale Festplatte zu einem zuverlässigen Device macht. Die Verwendung von Vandermonde-basierten Reed-Solomon Codes erlaubt neben der beliebig einstellbaren Fehlertoleranz eine optimale Platznutzung gemäß der Singleton-Grenze. Die Prototyp-Realisierung als Module für den Kernel des GNU/Linux Betriebssystems hat in einer Reihe von Konsistenz-, Funktions- und Stabilitätstests die Validität des Konzeptes gezeigt, insbesondere auch im Zusammenspiel mit Cluster-Dateisystemen wie dem MOSIX FS.

## Literatur

[BCH+01] S. Bethke, M. Calvetti, H. F. Hoffmann, D. Jacobs, M. Kasemann und D. Linglin. Report of the Steering Group of the LHC Computing Review. Bericht CERN/LHCC/2001-004

CERN/RRB-D 2001-3, European Organization for Nuclear Research (CERN), 2001.

[BDH03] L. A. Barroso, J. Dean und U. Hoelzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, Maerz-April 2003.

[BGW93] A. Barak, S. Guday und R. Wheeler. *The MOSIX Distributed Operating System, Load Balancing for UNIX*, Jgg. 672 of *Lecture Notes in Computer Science*. Springer, 1993.

[BLA00] P. T. Breuer, A. M. Lopez und Arturo G. Ares. The Enhanced Network Block Device. *Linux Journal*, May 2000.

[CER] CERN: European Laboratory for Particle Physics. http://www.cern.ch.

[GGL03] S. Ghemawat, H. Gobioff und S.-T. Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, Seiten 29–43, Bolton Landing, New York, USA, October 2003.

[Goo] Google Web Search Engine. http://www.google.com.

[HHL+00] D. R. Hankerson, D. G. Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. A. Rodger und J. R. Wall. *Coding Theory and Cryptography, The Essentials*. Pure and Applied Mathematics. Dekker, 2000.

[LHC] LHC:The Large Hadron Collider. http://web.cern.ch/lhc.

[LW03] V. Lindenstruth und A. Wiebalck. Verfahren und Vorrichtung zum Sichern von Daten bei mehreren unabhaengigen Schreib-Lese-Speichern. Als Patent eingereicht beim DPMA, amtl. Aktenzeichen 103 50 590.3, 2003.

[PD05] J. S. Plank und Y. Ding. Note: Correction to the 1997 Tutorial on Reed-Solomon Coding. *Software - Practice & Experience*, 35(2):189–194, February 2005.

[PGK88] D. A. Patterson, G. A. Gibson und R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seiten 109–116, Chicago, Illinois, USA, June 1988.

[Pla97] J. S. Plank. A Tutorial on Reed-Solomon Coding for Fault-tolerance in RAID-like Systems. *Software - Practice & Experience*, 9(27):995–1012, September 1997.

[TOP] TOP500, Die Top500 Supercomputer Seiten. http://www.top500.org.

[Wie05] A. Wiebalck. *ClusterRAID: Architecture and Prototype of a Distributed Fault-Tolerant Mass Storage System for Clusters*. Dissertation, Ruprecht-Karls-Universitaet Heidelberg, Juni 2005.

**Arne Wiebalck** wurde am 29. November 1974 in Bremerhaven geboren. Nach dem Abitur nahm er das Studium der Physik an der Ruprecht-Karls-Universität Heidelberg auf, das er im November 2000 mit einer Arbeit über Datenkompression für die Zeitprojektionskammer des ALICE Experiments abschloss. Am Lehrstuhl für Technische Informatik entwickelte er in seiner Promotion das ClusterRAID, eine Architektur und den Prototypen eines verteilten Massenspeichersystems für Cluster. Zur Zeit arbeitet er als Software-Entwickler im Bereich Lifecycle Management bei der SAP AG in Walldorf. Ab November 2006 wird er in der Linux und AFS Gruppe des IT Departments am Europäischen Kernforschungszentrum CERN nahe Genf eine neue Tätigkeit aufnehmen.