Fachhochschule Bonn-Rhein-Sieg

Studiengang Master of Computer Sciences SS2005

Access Behaviour: Eine Dateisystem-Auslastungsanalyse für groß skalierte wissenschaftliche Rechnenanwendungen

Seminararbeit zur Vorlesung „Paralele Systeme“
bei Prof. Dr. R. Berrendorf

Dipl. Inf. (FH) Martin Pöpping

25. Juni 2005

1 Einleitung

Wissenschaftliche parallele Anwendung haben einen sehr hohen Daten- und Rechenaufwand und benötigen ein Höchstmaß an Performance. Während neben der Prozesorrsleistung auch die Speicherkapazität in den letzten Jahren kostant anstieg, entwickelten sich die I/O-Systeme immer mehr zu einem Bottleneck. Daher verlangen wissenschaftliche parallele Anwendungen ein high-performance Support für das darunter liegende Dateisystem. Diese Seminararbeit beschäftigt sich mit der Analyse des Dateisystem-Workloads in groß skalierten wissenschaftlichen Rechenumgebungen. Um die Performance des I/O-Subsystems zu erhöhen und das entstandene Bottleneck zu reduzieren oder komplett zu beseitigen muss das Design von parallelen high-performance Dateisystemen optimiert werden. Für ein erfolgreiches Design eines Dateisystems ist ein umfassendes Verständnis des I/O-Workloads von high-performance Anwendungen notwendig. In den frühen und mittleren 90er-Jahren konzentrierte sich die Forschung auf die Charakterisierung von parallelen I/O-Workload-Mustern und bietete Einsicht auf das Design von parallelen Systemen. Das folgende Jahrzehnt brauchte signifikante Verbesserungen der Hardware (Prozessor, Speicher, Kommunikations-Verbindungen und I/O-Devices). Zur selben Zeit skalierten die Systeme um den steigenden Anforderungen von Rechenleistung und Speicherkapazität zu genügen, was auch die Erstellung von neuen wissenschaftlichen Anwendungen ermöglichte. Diese Änderungen motivierten Wang et al. die Charakteristiken von parallelen I/O-Workloads erneut zu untersuchen. Als Grundlage für diese Arbeit diente daher hauptsächlich das Paper „File System Workload Analysis For Large Scale Scientific Computing Applications“ [Wang u. a. 2004]. In diesem Paper werden I/O-Aktivitäten mit Hilfe von drei typischen parallelen wissenschaftlichen Anwendungen untersucht: Das Benchmarking-Programm *ior2* und den physikalischen Simulationen f1 (auf 343 Knoten) und m1 (auf 1620 Knoten). Dabei wurden neben dem statischen Dateisystem auch der dynamische I/O-Workload analysiert. Grundlage für das Ziel dieser Analyse waren folgende Fragen[Wang u. a. 2004]:

- Wie groß/ wie alt sind die Dateien?
- Wie viele Dateien werden geoffnet, gelesen und geschrieben?
  Wie groß sind diese?
- Wie häufig treten typische Dateisystem-Operationen auf?
- Wie oft senden Knoten I/O-Requets?
  Wie groß sind die angeforderten Dateien?
- Welche Formen von Lokalität gibt es?
  Wie könnte Caching nützlich sein?
- Werden Daten oft von Knoten geteilt?
  Wie sind die File-Sharing-Pattern?
- Wie gut nutzen die Knoten die I/O-Bandbreite?

2 Tracing-Methoden

Als Testumgebung wurde ein Großrechner mit mehr als 800 Dual-Prozessor Knoten verwendet. Das System arbeitet unter einem Linux-Kernel der Version 2.4.18. Als paralleles Dateisystem dient eine Entwicklungsversion von *Lustre Lite*[1] Der Rechner steht am Lawrence Livermore National Laboratory in Kalifornien[LLNL 2005].

2.1 Datensammlung

Das Tracing (Aufzeichnen) von I/O-Aktivitäten in großen verteilten Dateisystemen ist schwierig, da die Tracing-Verfahren das Systemverhalten stören können und die Ergebnisse verfälschen. Eine übliche Methode ist es ein Trace-Module zu entwickeln, dass spezifische I/O-Systemaufrufe abfängt. Ein dedizierter Knoten des Clusters sammelt alle Trace-Daten und speichert sie auf der lokalen Festplatte. Wang et al. wählten aus Zeitgründen jedoch einen einfacheren Ansatz. Sie verwendeten das Tracing-Tool *strace*[strace 2005] mit Tracing-Parametern, die auf dateibezogenene Systemaufrufe optimiert wurden. *Strace* ist ein System-Call-Tracer, der Traces von Systemaufrufen von anderen Prozessen oder Programmen ausgeben kann. Es setzt hierbei zwischen dem Kernel und der libc an, die von Programmen für Systemaufrufe verwendet wird (Abbildung 1).

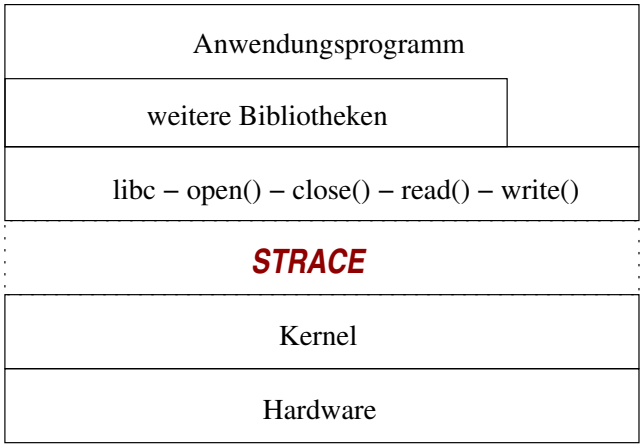


Abbildung 1: Strace setzt an den Kernel-Schnittstellen an, die auch von der libc verwendet werden

Das zu tracende Programm muss für *strace* nicht neu kompelliert werden, sondern kann auch in der Binärversion vorlegen. Die Trace-Daten wurden von *strace* auf die lokale Platte geschrieben. Dieser Ansatz bringt zwei Probleme mit sich. Zum einen zeichnet *strace* sehr viele Daten auf (paralleles Dateisystem, lokales Dateisystem, I/O-Aktivitäten etc.), so dass große Ansammlungen von Daten anfallen. Zum anderen verwendet *strace* das lokale Dateisystem für die Bufferung der Trace-Daten. Dieses Buffer-Schema arbeitet schlecht unter hohem I/O-Workload. Durch häufigen I/O-Zugriff auf die Trace-Daten kann

[1] Für weitere Informationen über Lustre sei auf die Seminararbeit von Martin Pelzer und Hanno Tersteegen verwiesen[Pelzer und Tersteegen 2005]

Gesamtanzahl der Knoten (IBM x355)	960
..... Rechen-Knoten	924
..... Login-Knoten	2
..... Gateway-Knoten	32
..... Metadata Server Knoten	2
Prozessoren pro Knoten (Pentium IV Prestonia)	2
Gesamtanzahl der Prozessoren	1920
Prozessorgeschwindigkeit (GHz)	2.4
Theoretisches Peak System-Performance (TFLops)	9.2
Speicher pro Knoten (GB)	4
Gesamter Speicher (TB)	3.8
Gesamter lokaler Plattenspeicher (TB)	115
Knoten-Verbindungen	Quadrics Switch

Tabelle 1: Die Daten des ASCI Linux Cluster

das Performance des Host-Systems beeinflusst werden. Wang et al. entschieden sich trotzdem für *strace*, da *strace* den mühsamen Vorgang der Sammlung von Daten auf eine inzelnes Shell-Skript vereinfacht. Nach der Meinung von Wang et al. spielen die oben genannten Probleme bei ihrer Simulation keine Rolle, da große I/O-Requests behandelt werden und relativ kurze Tracing-Perioden stattfinden. Auch bei häufigen I/O-Perioden tendiert die Anzahls der I/Os pro Knoten gegen durchschnittlich 10 Requests pro Sekunde. Das Buffern und Speichern der Daten hat daher nur einen geringen Einfluss auf das Performance des Systems. Die wissenschaftlichen Anwendungen, die für das Tracing der Daten verwendet werden, bestehen üblicherweise aus zwei verschiedenen Phasen, einer Berechnungsphase und einer I/O-Phase. Eine typische I/O-Phase dauert einige Minuten bis mehrere Stunden. Während dieser Phase generiert jeder Knoten einige hundert Kilobyte Trace-Daten, welche einfach im Speicher gebuffert werden können.

2.2 Anwendungen und Traces

Test-Cluster war der ASCI Linux Cluster am Lawrence Livermore National Laboratory[LLNL 2005], auf dem alle Trace-Daten gesammelt werden. Das Cluster ist mit seinen 960 Dual-Prozessor Knoten über einen Quadrics Switch verbunden. Zwei dieser Knoten laufen dediziert als Metadaten-Server und 32 Knoten werden als Gateways für den Zugriff auf das globale parallele Dateisystem verwendet. Tabelle 1 zeigt die Konfiguration und Hardware-Ausstattung des ASCI Linux Clusters. Im Julie 2003 sammelten Wang et al. Trace-Daten von dreier typischer paralleler wissenschaftlicher Anwendungen. Diese Tracing-Daten erreichten eine Gesamtgröße von mehr als 800 MB.

Die erste Testanwendung, *ior2* [ior2 2005], ist ein vom LLNL entwickeltes Benchmarking-Programm für parallele Dateisysteme, dass POSIX, MPIIO und HDF5-Interfaces verwendet und unter der GPL erhältlich ist. Es schreibt eine große Anzahl von Dummy-Daten in eine oder mehrere Dateien und liest diese dann anschließend wieder ein um die Korrektheit dieser Daten zu überprüfen. Diese Daten sind groß genug um den Caching-Effekt des Betriebssystems zu minimieren. Basierend auf drei verschiedenen Verwendungen von Dateien, wur-

den drei verschiedene Benchmark-Traces erstellt: *ior2-fileproc*, *ior2-shared* und *ior2-stride*. Alle drei Benchmarks liefen auf einem 512-Knoten-Cluster. *ior2-fileproc* wurde konfiguriert um eine individuelle Ausgabedatei pro Knoten zu erzeugen. *Ior2-shared* und *ior2-stride* verwendeten eine gemeinsame Daten für alle Knoten, wobei ior2-shared *zusammenhängenden* in die gemeinsame Datei schreibt, während *ior2-stride* Blöcke verschiedener Knoten in die gemeinsame Datei *nicht-zusammenhängend* schreibt. Die zweite Anwendung (*f1*) ist eine physikalische Simulation auf 343 Prozessoren. In dieser Anwendung sammelt ein einzelner Knoten eine große Menge von Daten in kleinen Stücken von den anderen Knoten. Eine kleine Anzahl von Knoten schreibt diese Daten anschließend in eine gemeinsame Datei. Die *f1*-Anwendung hat zwei I/O-Phasen. Eine *Restart-Phase*, bei der Lesezugriffe dominieren, wobei die zu lesenden Dateien von einer einzelnen Datei gelesen werden, die unabhängig voneinander auf jedem Knoten gespeichert sind. In der zweiten Phase, der *result dump*-Phase, werden hauptsächlich Dateien gespeichert. Die Traces werden im Folgenden *f1-restart* und *result-write* genannt. Die dritte Anwendung (*m1*) ist ebenfalls eine -auf 1620 Knoten laufende- physikalische Simulation. Diese Anwendung verwendet individuelle Output-Files für jeden Knoten. Wie bei der vorherigen Anwendung gibt es ebenfalls eine restart und eine result-dump Phase, die analog *m1-restart* und *m1-write* genannt werden.[Wang u. a. 2004]

2.3 Analyse

Für die Analyse werden die von *strace* erstellten Trace-Files aufbereitet, indem einige unwichtige System-Calls und -Signale herausgefiltert werden. Da jeder Knoten seine eigenen Trace-Records besitzt, liegen die Trace-Daten in mehreren hundert verschiedenen Dateien vor. In einer chronologischen Reihenfolge werden diese Dateien zu einer gemeinsamen Trace-Datei zusammengefasst. Zeitgeber für die chronologische Reihenfolge war der Quadrics-Switch des Clusters, der eine eigene Zeituhr besitzt, welche für die chronologische Reihenfolge verwendert werden konnte.

3 Charakteristik des Workloads

Mit Hilfe der strace-Daten wude der Workload des Clusters charakterisiert. Hierbei sollen die Fragen aus Abschnitt 1 über Verteilung, Größe und Lebenszeit der Dateien sowie Anzahl der Lese- und Schreib-Anforderungen u.a. charakterisiert werden.

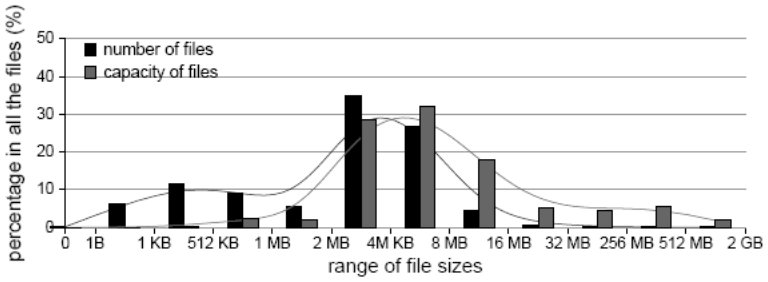
3.1 Dateiverteilung

Es wurden verteilte Dateien von 32 File-Servern gesammelt, die während der Laufzeit des Clusters verwendet wurden. Im Durchschnitt speicherte jeder File-Server 350.250 Dateien mit 1.04 TB Daten, was über 70 Prozent ihrer Kapazität entspricht. Tabelle 2 zeigt statistische Werte der Anzahl und Kapazität der Dateien auf diesen Servern. In Abbildung 2 wird die Verteilung der Dateien dargestellt. Diagramm (a) zeigt die Größe der Dateien durch Kapazität und Anzahl. Hieraus lässt sich erkennen, dass 80 Prozent der Dateien eine Größe zwischen 512 Kilobyte und 16 Megabyte haben und diese Dateien ca. 80 Prozent

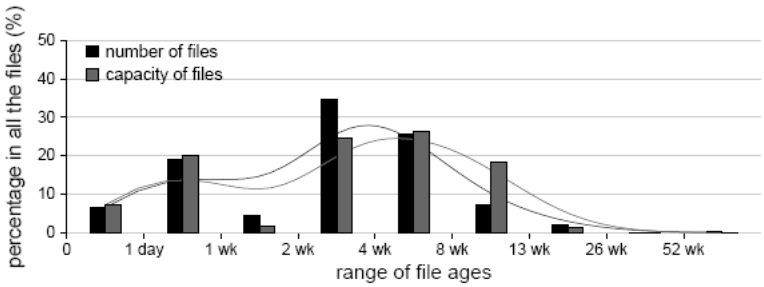
	Anzahl	**Kapazität**
Durchschnitt	305.200	1044.33 GB
Standardabweichung	75.760	139.66 GB
Mittelwert	305.200	1072.88 GB
Minimum	67.276	557.39 GB
Maximum	605.230	1207.37 GB

Tabelle 2: Die Daten des ASCI Linux Cluster

der Gesamtkapazität einnehnmen. Am auffälligsten sind die Dateien von 2 bis 8 Megabyte, die 61.7 Prozent aller Dateien und 60.5 Prozent aller Bytes in dem dargestellten Bereich ausmachen. Abbildung 2 (b) zeigt die Lebenszeit der Dateien in 9 Kateogrien zwischen 0-1 Tagen und 52 Wochen oder älter, wobei anzumerken ist, dass das Trace-System ein Jahr lief und daher keine Dateien die älter als 1 Jahr waren existierten. Wie aus dem Diagramm zu erkennen ist, lag die Lebenszeit von 60 Prozent der Dateien und 50 Prozent der Bytes zwischen 2 und 8 Wochen. 6.6 Prozent der Dateien und 7.3 Prozent der Bytes wurden nicht länger als einen Tag gespeichert.



(a) By File Sizes



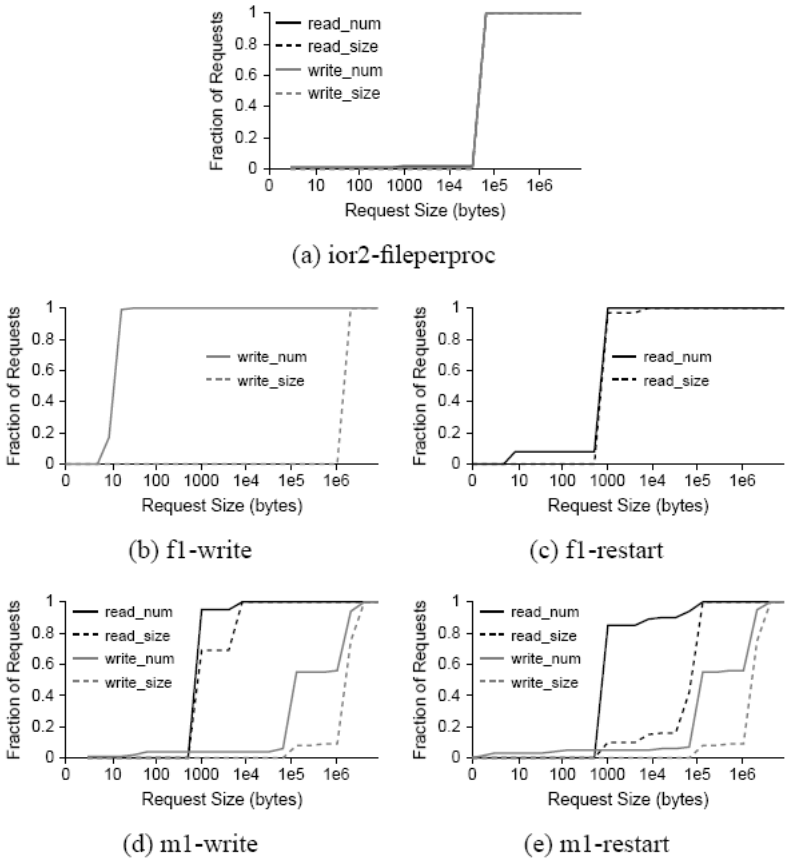
(b) By File Ages

Abbildung 2: Verteilung der Dateien anhand von Größe (a) und Alter (b)[Wang u. a. 2004]

3.2 I/O Request-Größe

Abbildung 3 zeigt die kummulative Verteilungsfunktion der Request-Größen und Request-Nummern. Da alle *ior2* Benchmarks dieselben Werte haben, wird

dieses Diagramm nur einmal dargestellt. Wie in Abbildung 3(a) zu sehen ist, hat *ior2* eine feste Request-Größe von 64 Kilobyte. Abbildung 3(b) zeigt die Schreib-Request Größenverteilung der result-dump Phase für das Programm *f1*. Alle Schreib-Requests sind kleiner als 16 Byte. Die gesamten I/O-Daten werden durch Requests mit größen von mehr als einen Megabyte übertragen. Hieraus ergibt sich ein übliches Muster für wissenschaftliche Anwendungen. Ein Master-Knoten sammelt kleine Datenteile bzw. Ergebnisse von den anderen Knoten und schreibt diese in Dateien. Die Diagramme (d) und (e) zeigen dieselbe Funktion in der restart und redsult-dump Phase des Programms *m1*. Die zwei Spitzen in der *writenum* Kurve zeigen zwei Hauptgrößen, 64 Kilobyte und 1.75 Megabyte. Mehr als 95 Prozent der Daten werden über große Requests übertragen, was auch aus (d) und (e) ersichtlich ist.



(a) ior2-fileperproc

(b) f1-write

(c) f1-restart

(d) m1-write

(e) m1-restart

Abbildung 3: Kummulative Verteilungsfunktion der I/O-Requests[Wang u. a. 2004]

3.3 I/O Access-Charakteristik

Abbildung 4 zeigt die Charakteristik des I/O-Zugriffs von *ior2*. Der *ior2-fileproc* Benchmark verwendet das one-file-per-node Dateimodell, das das beste Schreib-Performance bietet. Bis zu 150.00 Schreib-Requests (9 GB) pro Sekunde werden

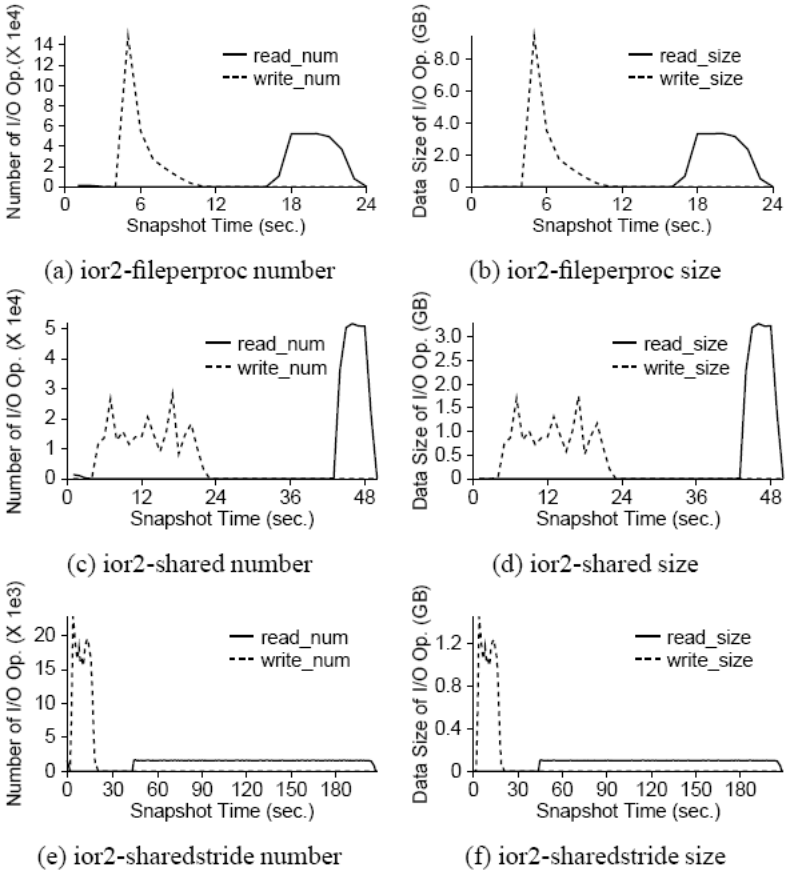


Abbildung 4: ior2 Benchmark-Ergebnisse der Requests[Wang u. a. 2004]

von den 512 Knoten generiert. Die *ior2-shared* und *ior2-stride* Benchmarks erreichen nur 25.000 Schreib-Requests (2 GB) pro Sekunde. Diese Benchmarks verwenden das shared-region und shared-stride Dateimodell. Wang et al. nehmen an, dass die Performance-Einbußen mit den darunter liegenden Dateikonsistenz-Protokoll zu tun haben. Die Gaps, die in Abbildung 4 zwischen der Schreib- und Lesekurve zu erkennen sind, reflektieren die aktuellen I/O-Zeiten. Offensichtlich bietet *ior2-fileproc* ein deutlich besseres Performance. Nur 10 Sekunden werden hier benötigt, wärend mehr als 20 Sekunden im shared-file Modell benötigt werden.*Ior2-stride* hat bei diesem Vergleich das schlechteste Lese-Performance. da nur 100 MB pro Sekunde gelesen werden können. Als Grund nennen Wang et al. das Datenlayout des Stride-Modells. Hier gibt es shared-file Limits für große sequentielle Leseoperationen, da die Daten nicht kontinuierlich geschrieben wurden.

Figure 3. I/O Requests over Time for *ior2* Benchmarks

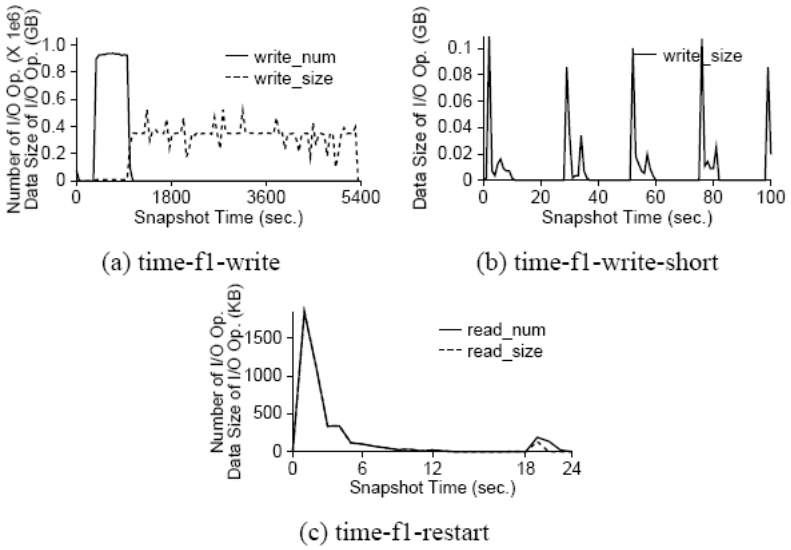


Abbildung 5: f1 Benchmark-Ergebnisse der Requests[Wang u. a. 2004]

Abbildung 5 zeigt die Access-Patterns von *f1*. Auffällig bei diesem Diagramm ist der Ausschlag der *write-num* Kurve. Dieser wird durch die Aktivitäten des Master-Knotens verursacht, der kleine Datenteile der anderen Knoten sammelt. Zu Spitzenzeiten werden hier fast eine Millionen Requests pro Sekunde gestellt. Während der restlichen Anwendung dominieren vor allem große Schreib-Requests der 48 Knoten die I/O-Aktivitäten. Die Requests werden in einem Burst-Mode angefordert, was aus Abbildung 5(b) gefolgert werden kann. Abbildung 5(b) ist ein Zoom in Abbildung 5(a). Aus Abbildung 5(c) lässt sich erkennen, dass in der Restart-Phase von *f1* die Lese-Requests dominieren.

Die I/O-Access Patterns der Anwendung *m1* werden in Abbildung 6 dargestellt. Daraus lässt sich sehr gut das Lese-Performance erkennen. Bis zu 28 GB pro Sekunde kann aufgrund der Größe der Dateien (1.6 MB bis 16 MB) durch die 1620 Knoten erreicht werden. Wie in Abbildung 6 zu erkennen ist, haben die Schreib-Kurven ähnliche Formen. Sie beginnen mit einem hohen Ausschlag

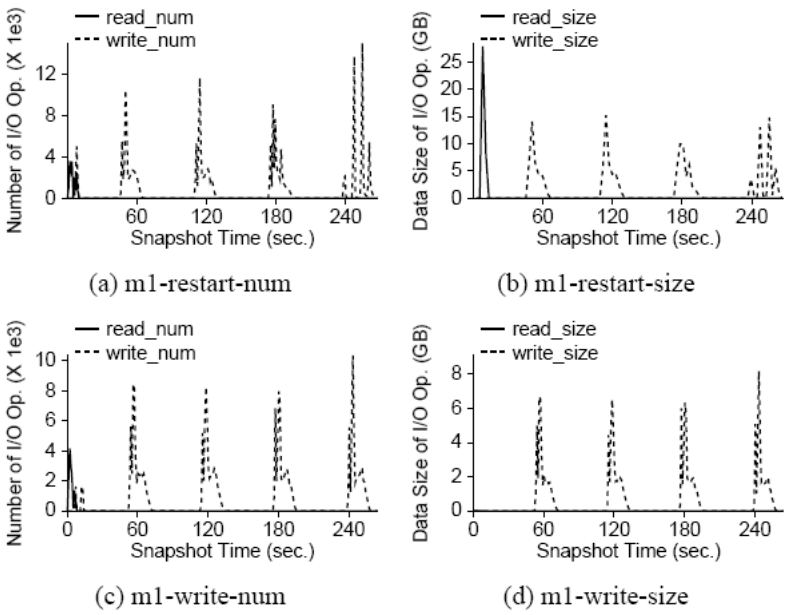


Abbildung 6: m1 Benchmark-Ergebnisse der Requests[Wang u. a. 2004]

und flachen dann in immer kleiner werdenen Ausschlägen ab. Eine mögliche Erklärung ist laut Wang et al. ist der Dateisystem-Buffer-Cache, der Schreib-Requests am Anfang des Schreibvorgangs abfängt. Ist der Buffer gefüllt, lässt die I/O-Rate deutlich nach.

3.4 Analyse der geöffnete Dateien

In allen Anwendungen werden Dateien meist als read/write oder read-only geöffnet. Die Tabellen 3 und 4 zeigen die geöffneten Dateien und Daten-Dateien. Mit Daten-Dateien sind hier Dateien gemeint, die Ergebnisse der Anwendungen speichern. Die Tabellen zeigen, dass es write-only Dateien nur in der Anwendung *m1* gibt. Die Daten-Dateien selbst werden, ausser bei *ior2* nur als read-only oder write-only geöffnet. Die Programme benötigen einige bis mehrere hundert Sekunden um eine Datei zu öffnen, wobei die Programme vor allem für Daten-Dateien deutlich länger brauchen. So gibt es bei *f1* beispielsweise Dateien, die bis zu 4 Gigabyte groß sind. Die Daten-Dateien der sonstigen Programme haben meist eine Größe von 6 bis 30 Megabyte. Die zu dem Anwendungsprogrammen gehörenden Dateien selbst sind meist zwischen 2 und 3 MB groß.

4 Zusammenfassung

Wang et al. untersuchten in Ihrer Studie Anwendungs-Traces eines Clusters mit mehreren hundert Knoten. Im Durchschnitt hat jede Anwendung nur ein oder zwei typische Request-Größen. Große Requests von mehreren hundert Kilobyte bis zu mehren Megabytes sind durchaus üblich. Obwohl bei einigen Anwendungen kleine Requests 90 Prozent aller Requests beanspruchen, werden die meisten

	Lesen/ Schreiben	Lesen	Schreiben
ior2	6,656	5,121	0
f1-write	3,871	6,870	718
f1-restart	3,773	6,179	0
m1-restart	17,824	22,681	12,940
m1-write	17,824	21,061	12,960

Tabelle 3: Statistik alle geöffneten Dateien

	Lesen/ Schreiben	Lesen	Schreiben
ior2	1,024	0	0
f1-write	98	10	34
f1-restart	0	343	0
m1-restart	0	1,620	12,940
m1-write	0	0	12,960

Tabelle 4: Statistik alle geöffneten Daten-Dateien

Daten bei großen I/O-Requests übertragen. Mehr als 65 Prozent aller Schreibzugriffe haben zwischenzeiten von einer Millisekunde bei den meisten Anwendungen. Alle Anwendungen zeigen explosionsartige[2] Zugriffsmuster. Dadurch, dass das selbe Benchmark auf verschiedenen Dateimodellen getestet wurde, konnte herausgefunden werden, dass der Schreibdurchsatz für eine individuelle Ausgabedatei für jeden Knoten fünf Mal größer ist, als wenn eine shared-Datei für alle Knoten verwendet würde. Dies zeigt, dass aktuelle Dateisysteme nicht gut für File-Sharing optimiert sind. In allen Anwendungen werden alle I/Os in einer kleinen Anzahl von Dateien gespeichert, die Zwischen- und Endergebnisse beinhalten. Diese Dateien werden in einer relativ lange Zeit von einigen bis mehre hundert Sekunden geöffnet. Hierbei wird eine große Menge von Daten transferiert.[Wang u. a. 2004]

Literatur

[ior2 2005] IOR2. *http://www.llnl.gov/icc/lc/siop/downloads/download.html*. 2005

[LLNL 2005] LLNL. *Lawrence Livermore National Laboratory - http://www.llnl.gov/*. 2005

[Pelzer und Tersteegen 2005] PELZER, Martin ; TERSTEEGEN, Hanno: *Seminararbeit Parallele Systeme: Lustre - Ein High-Performance-Dateisystem*. 2005

[strace 2005] STRACE. *http://strace.sourceforge.net*. 2005

[Wang u. a. 2004] WANG, F. ; XING, Q. ; HONG, B. ; BRANDT, S.A. ; MILLER, E.L. ; LONG, D.D.E.: *File System Workload Analysis For Large Scale Scientific Computing Applications*. 21st IEEE/ 12th NASA Goddard COnference on Mass Storage Systems and Technologies, 2004

[2] engl: bursty