

Berufsakademie Sachsen
Staatliche Studienakademie Leipzig

Evaluation und Optimierung der skalierbaren Storage-Lösung Ceph

Bachelorthesis

zur Erlangung der staatlichen Abschlussbezeichnung eines

Bachelor of Science

in der Studienrichtung Informatik

Eingereicht von: Jörg Broy

Mühlstr. 36

04317 Leipzig

cs14-1

Matrikelnr.: 5000553

Erstgutachter: Dr. Helmut Hayd

Max-Planck-Institut für

Kognitions- und Neurowissenschaften Leipzig

Zweitgutachter: Prof. Dr. Ingolf Brunner

Berufsakademie Sachsen

Staatliche Studienakademie Leipzig

Leipzig, 13. September 2017

Inhaltsverzeichnis

Abbildungsverzeichnis	6
Listingverzeichnis	9
Tabellenverzeichnis	10
Abkürzungsverzeichnis	12
1 Einleitung	13
2 Ceph	15
2.1 Begriffsklärungen	16
2.2 Speicherinterfaces	24
2.2.1 RADOS Block-Device	24
2.2.2 RADOS Object-Gateway	25
2.2.3 Ceph-Filesystem	25
2.3 Funktionsweise	27
2.3.1 Lage der Datenobjekte	27
3 Administration eines Ceph-Clusters	30
3.1 Pools und Placement-Groups	30
3.2 RADOS Block-Devices	32
3.3 Ceph-Filesystem	34
3.4 Schutz vor unberechtigttem Zugriff	35
4 Monitoring-Tools	38
4.1 collectd	38
4.2 graphite	39
4.3 ksysguard	40
5 Die Testsysteme	42
5.1 Commodity-Hardware-Cluster	42

5.1.1	Hardware	42
5.2	Profi-Hardware-Cluster	43
5.2.1	Hardware der Ceph-Server	44
5.2.2	Hardware der übrigen Server	44
5.3	Software	45
6	Benchmarking-Tools	49
6.1	Festplattengeschwindigkeit	49
6.1.1	Commodity-Hardware-Cluster	50
6.1.2	Profi-Hardware-Cluster	50
6.2	iperf	51
6.2.1	Commodity-Hardware-Cluster	52
6.2.2	Profi-Hardware-Cluster	53
6.3	Ceph-interner Benchmark	55
6.4	fio	56
7	Benchmarks nach Standardinstallation	59
7.1	Commodity-Hardware-Cluster	59
7.1.1	Ceph-interner Benchmark	60
7.1.2	RADOS Block-Device mit fio	60
7.1.3	Ceph-Filesystem mit fio	61
7.1.4	Zusammenfassung	62
7.2	Profi-Hardware-Cluster	62
7.2.1	Ceph-interner Benchmark	63
7.2.2	RADOS Block-Device mit fio	64
7.2.3	Ceph-Filesystem mit fio	64
7.2.4	Zusammenfassung	65
8	Ansatzpunkte für allgemeine Optimierungen	66
8.1	Filestore, SSDs, Replikation	66
8.1.1	Replikationszahl 1	68
8.1.1.1	Zwischenergebnis	69
8.1.2	Replikationszahl 2	70
8.1.2.1	Zwischenergebnis	72
8.1.3	Replikationszahl 3	72
8.1.3.1	Zwischenergebnis	74
8.1.4	Vergleich zur Konfiguration ohne SSDs	74

8.2	Bluestore, SSDs, Replikation	75
8.2.1	Replikationszahl 1	77
8.2.1.1	Zwischenergebnis	78
8.2.2	Replikationszahl 2	79
8.2.2.1	Zwischenergebnis	81
8.2.3	Replikationszahl 3	81
8.2.3.1	Zwischenergebnis	83
8.2.4	Vergleich zum Filestore mit SSDs	83
8.3	Bluestore, SSDs, Erasure-Coding	83
8.3.1	Anzahl Coding-Chunks 1	85
8.3.1.1	Zwischenergebnis	86
8.3.2	Anzahl Coding-Chunks 2	87
8.3.2.1	Zwischenergebnis	88
8.3.3	Anzahl Coding-Chunks 3	89
8.3.3.1	Zwischenergebnis	90
8.3.4	Vergleich zum Filestore mit SSDs	91
8.3.5	Vergleich zum Bluestore mit SSDs	91
9	Betrachtung von Lastszenarien	92
9.1	Konkurrierende Zugriffe	92
9.1.1	8 parallele Zugriffe	93
9.1.2	20 parallele Zugriffe	94
9.1.3	Auswertung	95
9.2	Datensicherheit versus Geschwindigkeit	96
9.2.1	Filestore und Bluestore im Vergleich	96
9.2.1.1	Interpretation	97
9.2.2	Speicherinterfaces im Vergleich	98
9.2.2.1	Interpretation	99
10	Auswirkungen von Schadszenarien	101
10.1	Ausfall einzelner OSDs	102
10.1.1	Auswirkungen beim Schreiben	103
10.1.2	Auswirkungen beim Lesen	104
10.2	Ausfall eines Ceph-Servers	105
10.2.1	Auswirkungen beim Schreiben	105
10.2.2	Auswirkungen beim Lesen	106

10.3 Zusammenfassung	107
11 Skalierbarkeit	108
11.1 Cluster verkleinern	108
11.2 Cluster vergrößern	111
11.2.1 Horizontale Skalierbarkeit	111
11.2.2 Vertikale Skalierbarkeit	113
11.3 Monitore und Metadata-Server	114
11.4 Zusammenfassung	115
12 Fazit und Ausblick	116
Literaturverzeichnis	118
Anhang A Listings	121
Anhang B CC-Lizenz	128
Selbstständigkeitserklärung	129

Abbildungsverzeichnis

2.1	Architektur eines Ceph-Clusters [1, S. 40]	16
2.2	OSD [1, S. 44]	17
2.3	OSD Journal [1, S. 45]	17
2.4	Placement-Groups [1, S. 68]	18
2.5	Pools	19
2.6	CRUSH-Lookup [1, S. 64]	20
2.7	Aufbau eines OSD im Bluestore [2]	23
2.8	Snapshots und Klone [1, S. 114]	25
2.9	CephFS [1, S. 57]	26
4.1	graphite Dashboard zur Überwachung limitierender Faktoren. Grafiken eines Ceph-Servers stehen untereinander.	40
4.2	ksysguard zur Überwachung limitierender Faktoren; Grafiken eines Ceph-Servers stehen untereinander.	41
6.1	Durchschnittliche Schreib- u. Leseraten der Festplatten (je 3 pro Ceph-Server)	50
6.2	Durchschnittliche Schreib- u. Leseraten der Festplatten (je 12 pro Ceph-Server)	51
6.3	Testanordnung für iperf-Test	52
6.4	Netzwerkgeschwindigkeit bei sukzessiver Messung	53
6.5	Netzwerkgeschwindigkeit bei paralleler Messung	53
6.6	Testanordnung für iperf-Test	54
6.7	Netzwerkgeschwindigkeit bei sukzessiver Messung	54
6.8	Netzwerkgeschwindigkeit bei paralleler Messung	55
7.1	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	60
7.2	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host	61
7.3	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	61
7.4	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	63
7.5	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host	64

7.6	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	64
8.1	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	68
8.2	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	69
8.3	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	69
8.4	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	71
8.5	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	71
8.6	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	72
8.7	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	73
8.8	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	73
8.9	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	74
8.10	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	77
8.11	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	78
8.12	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	78
8.13	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	80
8.14	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	80
8.15	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	81
8.16	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	82
8.17	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	82
8.18	Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host	83
8.19	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	86
8.20	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	86
8.21	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	88
8.22	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	88
8.23	Schreib-/Leseraten, verschiedene Blocksize, 64 Threads	90
8.24	Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host . . .	90
9.1	Schreib-/Leseraten, bei je 2 gemounteten Block-Devices pro Host	93
9.2	Schreib-/Leseraten, bei je 2 gemounteten Verzeichnissen pro Host	94
9.3	Schreib-/Leseraten, bei je 5 gemounteten Block-Devices pro Host	94
9.4	Schreib-/Leseraten, bei je 5 gemounteten Verzeichnissen pro Host	95
9.5	RADOS-bench im Vergleich der Konfigurationen	96
9.6	Block-Device im Vergleich der Konfigurationen	97
9.7	Ceph-Filesystem im Vergleich der Konfigurationen	97
9.8	Vergleich der Tests mit Konfiguration: Filestore, SSDs, Replikation	98
9.9	Vergleich der Tests mit Konfiguration: Bluestore, SSDs, Replikation	99
9.10	Vergleich der Tests mit Konfiguration: Bluestore, SSDs, Erasure-Coding . .	99

10.1	Veränderung der Schreibleistung bei 3 gleichzeitig ausgefallenen OSDs . . .	103
10.2	Veränderung der Schreibleistung bei 3 sich wiedereingliedernden OSDs . .	103
10.3	Veränderung der Leseleistung bei 3 gleichzeitig ausgefallenen OSDs	104
10.4	Veränderung der Leseleistung bei 3 sich wiedereingliedernden OSDs	104
10.5	Veränderung der Schreibleistung bei einem komplett ausgefallenen Ceph- Server	105
10.6	Veränderung der Schreibleistung bei einem sich wiedereingliedernden Ceph- Server	105
10.7	Veränderung der Leseleistung bei einem komplett ausgefallenen Ceph-Server	106
10.8	Veränderung der Leseleistung bei einem sich wiedereingliedernden Ceph-Server	107

Listingverzeichnis

2.1	Ceph-Datenobjekt erzeugen	28
2.2	Beteiligte OSDs ermitteln	28
2.3	Lage der OSDs ermitteln	28
2.4	Ceph-Datenobjekt finden	29
3.1	Erstellung und Löschung eines Pools	32
3.2	Vorbereiten des Clients	33
3.3	Einbinden eines Block-Devices	33
3.4	Löschen eines Block-Devices	33
3.5	Erstellen eines Ceph-Filesystems	34
3.6	Einbinden von CephFS mit Kernelmodul	35
3.7	Einbinden von CephFS mit Fuse	35
3.8	Erstellen eines Nutzers	36
3.9	Veränderung der Rechte eines Nutzers	36
4.1	Installation von collectd	38
4.2	Installation von graphite	39
5.1	Einrichten eines Nutzers für Ceph	45
5.2	SSH-Key erstellen und verteilen	46
5.3	SSH-Tuning	46
5.4	Installation von ceph-deploy auf dem Admin-Server	46
5.5	Installation von Ceph	47
5.6	Hinzufügen von OSDs	48
5.7	Hinzufügen von Monitoren und MDS	48
6.1	Ermitteln der Schreib- und Lesegeschwindigkeit einer Festplatte mit bonnie++	49
6.2	Ceph-interner Benchmark	55
6.3	fio Beispiel für CSV-Ausgabe	57
6.4	fio-Kommando für Ceph-Filesystem und Lesen von RADOS Block-Devices	58
6.5	fio-Kommando für das Schreiben auf RADOS Block-Devices	58
8.1	Einrichten von OSDs mit separaten Journalen	67

8.2	Erstellen eines Pools mit der Replikationszahl eins	67
8.3	Installation von Kernel 4.11 und luminous	76
8.4	Beispiel für Einrichtung zweier Bluestore OSDs samt Metadatenbanken und WALs	76
8.5	Erzeugen eines Erasure-Coding Pools für das RADOS Block-Device	85
10.1	Ausschalten von OSDs und Wiederhinzufügen	102
11.1	Status des Clusters in voller Ausbaustufe	109
11.2	Baumstruktur des Clusters in voller Ausbaustufe	110
11.3	Befehle zum Entfernen eines OSDs	110
11.4	Befehle zum Entfernen eines Ceph-Servers	111
11.5	Clusterstatus nach Entfernung von osceph6	111
11.6	Hinzufügen eines Cephserver mit 6 OSDs (mit SSD fürs Journal)	112
11.7	Clusterstatus nach Hinzufügen eines Cephserver und 6 OSDs	112
11.8	Baumstruktur des Clusters nach Hinzufügen eines Cephserver und 6 OSDs	113
11.9	Hinzufügen von 6 OSDs (mit SSD fürs Journal) zu einem Ceph-Server . . .	113
11.10	Clusterstatus nach Hinzufügen von 6 OSDs	114
11.11	Baumstruktur des Clusters nach vertikaler Skalierung	114
11.12	Erstellen zusätzlicher Monitore und Metadata-Server	115
A.1	Script zur Installation und Konfiguration von collectd	121
A.2	Ausgangskonfigurationsdatei von collectd	125
A.3	CRUSH placement for Object x [3]	127

Tabellenverzeichnis

3.1	Recovery bei Ausfall eines OSDs	31
3.2	Berechnung der Anzahl an Placement-Groups	32
5.1	Technische Daten des Commodity-Hardware-Clusters	43
5.2	Technische Daten der Ceph-Server des Profi-Hardware-Clusters	44
5.3	Technische Daten der übrigen Server des Profi-Hardware-Clusters	44
6.1	Kommandozeilenparameter von fio [4]	57

Abkürzungsverzeichnis

BlueFS	Bluestore File System
BS	Bluestore
CephFS	Ceph File System
CHWC	Commodity-Hardware-Cluster
CRUSH	Controlled Replication Under Scalable Hashing
EC	Erasure-Coding
fio	Flexible I/O Tester
FS	Filestore
MDS	Meta Data Server
MGR	Management Server
OSD	Object-Storage-Device
PG	Placement-Group
PHWC	Profi-Hardware-Cluster
RADOS	Reliable Autonomic Distributed Object Store
RBD	RADOS Block-Device
WAL	Write Ahead Log

Kapitel 1

Einleitung

Ziel dieser Arbeit ist die Evaluation und Optimierung der skalierbaren Storage-Lösung Ceph. Evaluation bedeutet, dass mithilfe von Werkzeugen wie fio, bonnie++ und iperf gemessen wird, welche Schreib- und Leseraten ein Ceph-Cluster unter verschiedenen Bedingungen erreichen kann. Diese Geschwindigkeiten sollen erhöht werden, indem der Rechnerverbund nach bestimmten Kriterien optimiert wird.

Im zweiten Kapitel wird das Storage-System Ceph in seinen Grundzügen theoretisch umrissen. Dabei wird auf die zum Verständnis wesentlichen Begriffe eingegangen und anschließend werden die drei verfügbaren Speicherarten bzw. -interfaces Block-Device, RADOS Object-Gateway und Ceph-Filesystem erklärt. Im weiteren Verlauf dieser Arbeit wird sich auf Block-Devices und das Ceph-Filesystem konzentriert werden, da das RADOS Object-Gateway für den Auftraggeber dieser Studie nicht weiter von Interesse ist. Im Abschluss der allgemeinen Betrachtungen wird die Funktionsweise von Ceph kurz umrissen, um einen Eindruck zu vermitteln, wie das Speichern und Lesen im Innern eines Ceph Clusters funktioniert.

Im dritten Kapitel werden grundlegende Operationen dargestellt, mit deren Hilfe sich Pools anlegen und Block-Devices in Clients einbinden lassen. Desweiteren wird gezeigt, wie das Ceph-Filesystem genutzt werden kann und man den Ceph-Cluster vor unberechtigtem Zugriff mithilfe von Nutzern und Schlüsseln schützt.

Im vierten Kapitel werden die verwendeten Monitoringprogramme vorgestellt, da es während der Messungen von Schreib- und Lesegeschwindigkeiten nötig ist, die zugrundeliegenden Systeme permanent zu überwachen. Parameter wie CPU-Auslastung, Hauptspeicherverbrauch und Netzwerkdurchsatz werden von den Programmen „collectd“ und „graphite“ erhoben, aufgezeichnet und visualisiert. Des Weiteren wird das Programm „ksysguard“ zur Live-Beobachtung verwendet.

Im fünften Kapitel wird auf die Testsysteme eingegangen. Ein wesentliches Merkmal Ceph's ist seine Anspruchslosigkeit, was Hardware betrifft. Es läuft sowohl auf handelsüblicher Technik, wie sie im Consumerbereich eingesetzt wird, sogenannter Commodity-Hardware, als auch auf schnellen Servern, wie sie im Profisegment anzutreffen sind. Um dieses Feature zu zeigen, werden die Bestandteile der beiden Cluster im Kapitel Testsysteme betreffs Hard- und Software erläutert.

Im sechsten Kapitel werden die Verfahren erläutert, mit denen die Geschwindigkeitsmessungen durchgeführt wurden. „Iperf“ wird für die Netzwerkgeschwindigkeit und „bonnie++“ für die reinen Schreib- und Leserate der verbauten Festplatten verwendet. Da Ceph ein eigenes Werkzeug zum Benchmarking mitbringt, wird auch dieses erläutert und zum Einsatz gebracht. Schließlich kommt die Rede auf „fio“, den Flexible IO Tester, welcher in der Hauptsache für die Ermittlung der Schreib-/ Leseperformance benutzt wurde.

Im siebten Kapitel werden die beiden Cluster in der Standardkonfiguration getestet, in der sie sich nach einer Grundinstallation Ceph's befinden. Dabei werden Profi- und Commodity-Hardware-Cluster miteinander verglichen.

Im achten Kapitel werden verschiedene Optimierungen vorgenommen und jeweils getestet, welche Auswirkungen sie auf die Performance des Clusters haben. Ab diesem Kapitel wird nur noch der Profi-Hardware-Cluster betrachtet.

Im neunten Kapitel schließen sich Betrachtungen an, den Cluster für bestimmte Lastszenarien, wie bspw. viele konkurrierende Zugriffe, hohe Datensicherheit und maximale Geschwindigkeit zu optimieren. Es wird versucht, zu validieren, ob diese Verbesserungen einen erheblichen Effekt erzielen können.

Im zehnten Kapitel wird der Ausfallsicherheit des Clusters nachgegangen, in dem der Defekt einzelner Festplatten wie auch ganzer Server simuliert wird.

Im elften Kapitel wird auf die horizontale und vertikale Skalierbarkeit eingegangen. Dabei wird untersucht, ob es einfach möglich ist, den Cluster mit neuen Servern zu erweitern bzw. diese zu entziehen, als auch einen einzelnen Server mit zusätzlichen Festplatten auszustatten bzw. etwaige Platten zu entnehmen, ohne den Betrieb des Clusters unterbrechen zu müssen.

Im zwölften Kapitel wird resümiert, ob die wesentlichen Anliegen der Arbeit erreicht werden konnten und in welchem Maße die versprochenen Leistungsmerkmale zutreffen oder nicht.

Kapitel 2

Ceph

Ceph, dessen Name sich von Cephalopoda (Kopffüßler) [5] ableitet, da es mit diesem die Redundanz von lebenswichtigen Gliedmaßen bzw. Komponenten gemein hat, ist ein verteiltes System zur Speicherung und Bereitstellung von Daten. Es speichert diese in Form von Objekten (Object Storage [6]), die es nach bestimmten Regeln über einen Cluster aus beliebig vielen Servern mit frei einstellbarer Redundanz verteilt.

Das im Jahre 2003 von Sage Weil an der Universität von Kalifornien als PhD-Arbeit begonnene Ceph, wurde ab 2012 vom eigens gegründeten Unternehmen Inktank zur Produktreife geführt und nach einer Übernahme seit 2014 von Red Hat federführend weiterentwickelt. Aufgrund der von Ceph versprochenen Features [1, S. 8], die es zu einem universellen Speichersystem, einer „Unified-Storage-Solution“ machen sollen, ist es vor allem für Provider interessant, die sich mit einer ständig wachsenden Flut von Daten konfrontiert sehen, welche permanent, ausfallsicher und schnell gespeichert bzw. gelesen werden müssen. Die versprochenen Features sind:

- Es gibt keinen „Single-Point-of-Failure“.
- Es ist „Software-defined“ und Open Source (LGPL).
- Es ist auf heterogener Commodity-Hardware lauffähig.
- Es skaliert horizontal und vertikal.

Skalierbarkeit, Ausfallsicherheit und Lauffähigkeit auf handelsüblicher „Commodity-Hardware“ werden in dieser Arbeit Gegenstand von Überprüfungen sein.

Ein weiterer Vorteil von Ceph liegt darin begründet, dass seit 2010 im Linux-Kernel entsprechende Module existieren, die eine Kommunikation zwischen Linux und Ceph ermöglichen. So gibt es Schnittstellen für die Anbindung von Ceph-Block-Devices [7, Kap.2, S.2] und für das ab 2016 verwendbare CephFS - das Ceph-Filesystem.

2.1 Begriffsklärungen

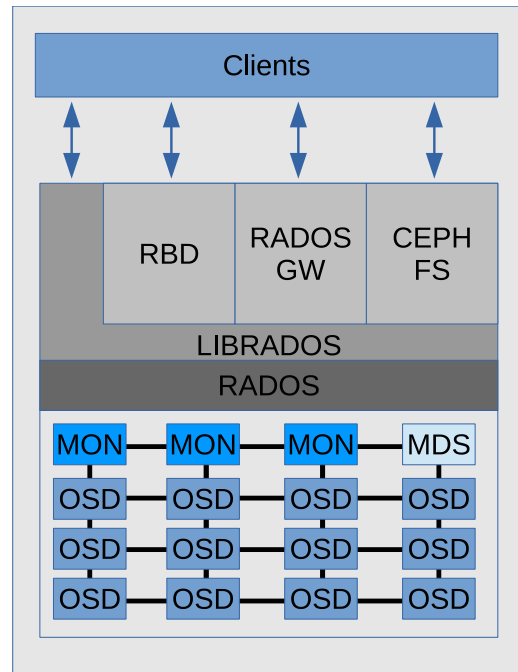


Abbildung 2.1: Architektur eines Ceph-Clusters [1, S. 40]

RADOS - Bei Ceph handelt es sich um ein Software-defined-Storage System [7, Kap.2, S.2]. Der Software-Layer, welcher die entsprechenden Funktionalitäten im Cluster bereitstellt, heißt RADOS (Reliable Autonomic Distributed Object-Store). Er stellt die Schicht dar, die der Hardware am nächsten liegt und ist das eigentliche Herz Ceph's. Er kümmert sich um die Umwandlung von Dateien in Objekte, deren Verteilung im Cluster und die Sicherstellung ihrer Konsistenz.

Objekte - Die fundamentale Einheit im Ceph-Cluster heißt Objekt [1, S. 59f]. In einem Ceph-Verbund werden die Daten in Form von Objekten gespeichert. Das bedeutet streng genommen, dass alle Dateien, die ein Client in Ceph speichern will, zunächst in Objekte umgewandelt werden müssen. Ein Objekt besteht aus einer einmaligen ID, einem beliebig großen Daten- und einem Metadatenteil. Entsprechend des eingestellten Replikationslevels werden die Objekte während eines Speichervorganges redundant im Cluster verteilt und gespeichert. Dabei ist zu beachten, dass ein Objekt nur ein logisches Element ist, das lediglich für Ceph diese Bedeutung hat. Die Ceph-Server selbst sind ganz normale Linux-Systeme, deren Festplatten Ceph zum Speichern dienen. Die Objekte liegen darin als Dateien in gewöhnlichen Dateisystemen. Objekte stellen sie nur für Ceph dar.

OSD - Bei einem „Object-Storage-Device“ [1, S. 43-47] handelt es sich im einfachsten Fall um ein Laufwerk eines Servers, das als Ganzes in den Ceph-Cluster aufgenommen und

zur Speicherung von Datenobjekten benutzt werden kann. Allerdings können auch einzelne Partitionen und Verzeichnisse und damit Teilbereiche eines Laufwerkes als OSD dienen. Für jedes OSD läuft ein OSD-Daemon auf dem Server, der den Datenträger enthält. Wie schon erwähnt, werden auf den OSDs die Datenobjekte des Clusters gespeichert. Die logischen Objekte Ceph liegen als Dateien im Dateisystem der OSDs, die ganz normal im Linux des Servers gemountet sind. Als Dateisysteme stehen zur Auswahl: btrfs, XFS und ext4. Für andere Dateisysteme besteht keine Unterstützung.

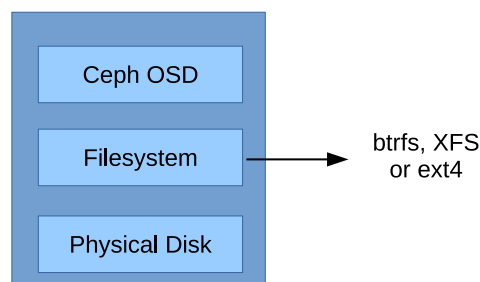


Abbildung 2.2: OSD [1, S. 44]

Jedes OSD verfügt über ein spezielles Journal, das auf einem separaten oder dem gleichen Laufwerk liegen kann. In diesem Journal werden alle Objekte zwischengespeichert, bevor sie endgültig auf die Festplatte des OSDs geschrieben werden. Der Vorteil liegt darin, dass viele kleine Objekte gruppiert werden können und so beim Schreiben Zeit gespart und einer Fragmentierung des Speichergerätes entgegengewirkt werden kann.

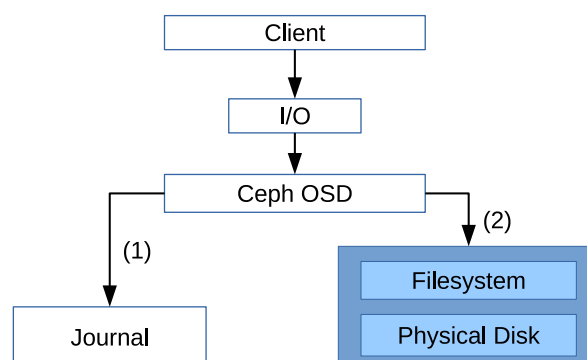


Abbildung 2.3: OSD Journal [1, S. 45]

Placement-Group - Placement-Groups (PGs) sind logische Gruppen von Objekten [1, S. 68]. Ihr Zweck besteht darin, die Ceph-Objekte besser handhabbar zu machen. Statt alle Objekte seines Speicherbereiches gleichzeitig verwalten zu müssen, etwa wenn lesend darauf zugegriffen werden soll, gestatten es die Placement-Groups dem OSD-Daemon, nur noch

auf Teilmengen davon zuzugreifen und die CPU und den RAM des Servers zu entlasten. Der Name Placement-Group selbst verdeutlicht, dass alle Objekte dieser Gruppe auf ein und demselben OSD gespeichert bzw. „platziert“ sind.

Jedes Objekt ist eineindeutig einer PG zugeteilt. Sie werden je nach Replikationslevel auf mehrere OSDs verteilt. Bei einem Replikationslevel von drei existiert eine PG bspw. auf drei Ceph-Servern gleichzeitig. Fällt ein OSD aus, werden die dort liegenden PGs mithilfe der noch verfügbaren PGs auf den verbliebenen zwei OSDs auf einem neuen OSD kreiert, sodass wieder drei Repliken von jeder PG und damit jedem Objekt existieren.

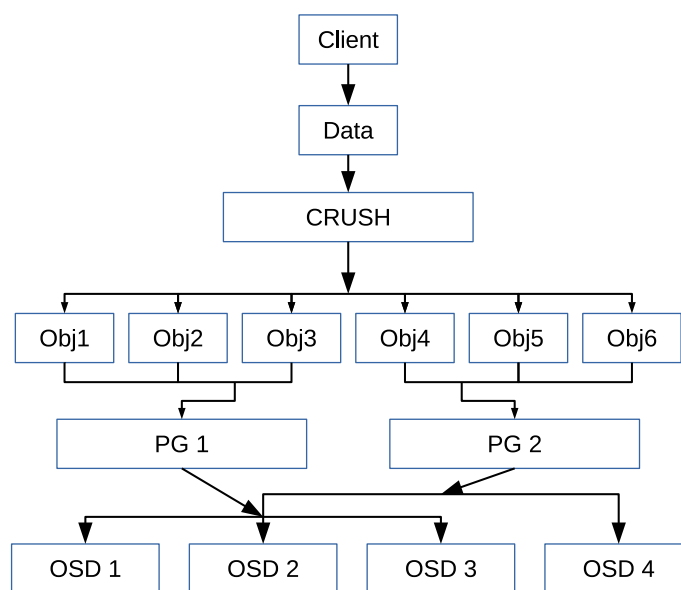


Abbildung 2.4: Placement-Groups [1, S. 68]

Pool - Ein Pool ist eine weitere logische Einheit und eine Ebene höher angesiedelt als die PGs [8]. Der gesamte Speicher eines Ceph-Clusters ist in Pools aufgeteilt, in denen sich die PGs befinden. Sie sind den Pools eineindeutig zugeordnet. Auf Poolebene werden bspw. Replikationslevel, Anzahl der PGs oder Erasure-Coding eingestellt, d.h. beim Erstellen eines Pools kann man diese Merkmale als Parameter übergeben.

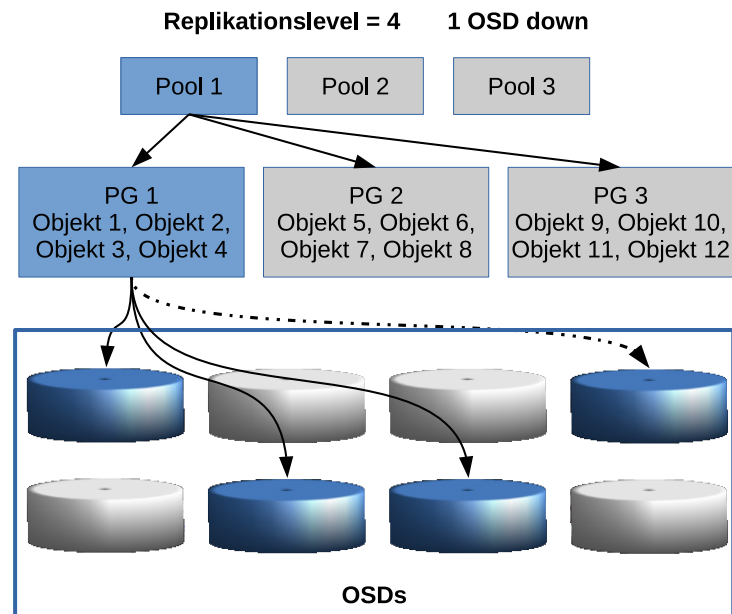


Abbildung 2.5: Pools

Replikation - Ein Ceph-Cluster soll Daten ausfallsicher bewahren, daher ist es nötig, Redundanz zu schaffen. Von jedem Objekt existiert eine frei definierbare Anzahl von Repliken. Diese Zahl regelt das Replikationslevel bzw. die Replikationszahl. Die Kopien sind dabei so über den Cluster verteilt, dass sie in verschiedenen Ausfallzonen liegen, die wiederum in den sogenannten CRUSH-Maps definiert werden können. Pools können als Replikationspools oder als Erasure-Coding Pools erstellt werden. Das Erasure-Coding wird weiter unten erklärt.

CRUSH - „Controlled Replication Under Scalable Hashing“ ist der Name eines Algorithmus, der für Ceph die Lage der Datenobjekte im Cluster berechnet [1, S. 62f]. Traditionelle Dateisysteme nutzen zumeist Metadaten, die in Lookup-Tabellen (bspw. FAT, Inode-Table) gespeichert sind, um die Lage der Daten auf einer Festplatte zu ermitteln. Ceph geht hier andere Wege, um zwei Probleme von Lookup-Tabellen zu vermeiden:

- Die Lookup-Tabellen sind die wichtigsten Daten. Wenn sie bei einem Ausfall zerstört werden, sind auch die gespeicherten Daten unzugänglich. Dieser Point-of-Failure wird durch CRUSH umgangen, da es keine Lookup-Tabellen gibt, sondern CRUSH die Lage der Daten bei jedem Request berechnet.
- Des Weiteren wäre es problematisch, Lookup-Tabellen für Daten in sehr großen Mengen vorzuhalten, etwa wenn Daten im Exabyte-Bereich vorliegen. Die Tabellen müssten dann selber gigantisch groß sein und wären daher speichertechnisch und wiederum ausfallbedingt problematisch.

Ein CRUSH-Lookup funktioniert folgendermaßen: Wird ein Schreibvorgang eingeleitet, oder sollen Daten gelesen werden, wird die Lage der Datenobjekte im Cluster von CRUSH berechnet. Der Vorgang des Berechnens der Lage im Cluster heißt CRUSH-Lookup. Er sei im Folgenden skizziert. Der ausführliche Code aus Sage Weils diesbezüglichem Paper [3] befindet sich im Anhang in Listing A.3 auf Seite 127.

1. Die Nummer der PG, in der das Objekt gespeichert ist oder gespeichert werden soll, wird per Hash-Funktion (siehe Anhang Listing A.3 auf Seite 127) berechnet. Eingabeparameter sind die eindeutige Objekt-ID, die Anzahl der PGs des betreffenden Pools und die ID des betreffenden Pools. Das Ergebnis der Berechnung ist die Nummer der PG, die im nächsten Schritt weiterverarbeitet wird.
2. Im zweiten Schritt wird berechnet, auf welchen OSDs die, im ersten Schritt bestimmte, PG gespeichert ist. Dies geschieht durch die eigentliche CRUSH-Funktion. Als Eingabeparameter erhält diese die, im ersten Schritt berechnete, PG-Nummer sowie den Status des gesamten Clusters und die CRUSH-Regeln, die in der Cluster-Map auf den Monitoren gespeichert sind. Das Ergebnis der Berechnung ist eine Gruppe von OSDs, auf denen die berechnete PG verortet ist. Nachdem die Lage der PG bekannt ist, kann das Objekt gespeichert oder gelesen werden.

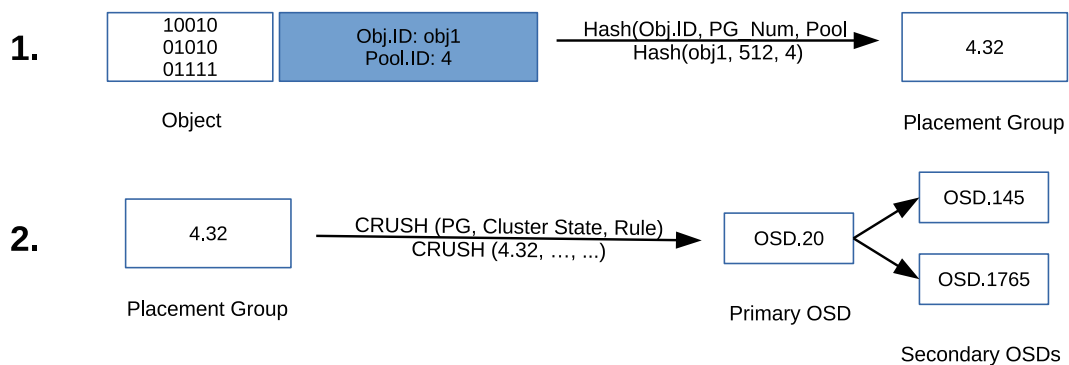


Abbildung 2.6: CRUSH-Lookup [1, S. 64]

Der Vorzug von CRUSH liegt darin, dass keine Lookup-Tabellen gespeichert werden müssen und damit deren Ausfall kein Point-of-Failure darstellen kann. Außerdem skaliert CRUSH und Ceph dadurch beliebig, da es keine nicht mehr ausreichend großen Lookup-Tabellen geben, die beim Anlegen, wie im Falle der Inode-Tabellen, in ihrer Größe vordefiniert und damit limitiert sind. Alles, was ein Client benötigt, um seine Daten im Ceph-Cluster zu finden, ist eine aktuelle Cluster-Map, die er von einem Monitor bekommt.

Ein weiterer Vorteil CRUSHs liegt darin, dass der CRUSH-Lookup auf dem Client geschieht, wodurch die CPUs des Clusters geschont und für andere Berechnungen, wie etwa das Erasure-Coding, frei gehalten werden können. Da heutzutage jeder Client, sei es Workstation oder Smartphone, über genügend Rechenleistung verfügt, sollte dieser Umstand kein Problem darstellen. Auch dieses Feature trägt dazu bei, dass Ceph beliebig skaliert.

In Sage Weils Paper zum Thema CRUSH [3] beziffert er die Komplexität und damit Laufzeit des Algorithmus mit $O(\log n)$ für einen Cluster mit n OSDs. Das bedeutet, der Algorithmus skaliert sehr gut und große Anzahlen von OSDs wirken sich höchstens logarithmisch auf die Laufzeit aus.

Um dies kurz an einem Beispiel zu verdeutlichen, seien lineare und logarithmische Komplexität miteinander verglichen. Bei 1.000.000 OSDs in einem sehr großen Cluster würde sich eine lineare Laufzeit von 1.000.000 Schleifendurchläufen zur Bestimmung der gewünschten OSDs ergeben. Bei logarithmischer Laufzeit dagegen ergeben sich bei einer Basis von bspw. 10 höchstens 6 Schleifendurchläufe und damit ein sehr ökonomisches Ergebnis.

Monitor - Da Ceph ein verteiltes System ist, müssen die Angaben über die Bestandteile dieses Systems an festgelegten Stellen aufbewahrt und zugänglich sein. Diese Rolle übernehmen die Monitore mit ihren entsprechenden Daemons [7, Kap. 7, S. 5f]. Sie halten die Cluster-Map und die Logdateien des gesamten Clusters vor. Im Speziellen besteht die Cluster-Map aus mehreren Teil-Maps, welche Informationen über die Monitore selbst, die angeschlossenen OSDs und die Placement-Groups, Metadata-Server und über CRUSH beinhalten. In einem Ceph-Cluster muss mindestens ein Monitor installiert sein. Um einen Single-Point-of-Failure zu vermeiden, sollten es aber mindestens drei sein. Da es vorkommen kann, dass aufgrund von Latenzen oder Hardwareausfällen ein Monitor nicht mehr die aktuellsten Daten über den Cluster bereithält, muss immer eine Mehrheit (Quorum) von Monitoren darüber entscheiden, welche Cluster-Map gerade ausschlaggebend ist. Von daher wird empfohlen, eine ungerade Anzahl von Monitoren zu installieren, da so leichter Mehrheiten gebildet werden können. Des Weiteren kann damit Split-Brain-Problemen vorgebeugt werden. D.h. falls der Cluster in zwei Teile zerfällt, weil evtl. Netzwerkprobleme bestehen, können Daten nicht fälschlicherweise in beiden Teilen geschrieben werden, da mindestens einer davon eine Anzahl von Monitoren enthält, die keiner Mehrheit aller bekannter Monitore entspricht.

MDS - Die Metadata-Server sind ausschließlich für das Ceph-Filesystem relevant [8]. Sie halten für das noch zu besprechende Ceph-eigene Dateisystem Metadaten vor, um schnellere Dateisystemoperationen, wie „ls“, „mkdir“ oder „find“ zu ermöglichen. Der MDS selbst

speichert dabei keine Metadaten. Er lädt sie nur aus dem entsprechenden Metadatenpool des Ceph-Clusters in seinen Hauptspeicher. Dadurch wird erreicht, dass ein ausgefallener MDS von einem anderen ersetzt werden kann, da die Metadaten genau wie die übrigen Daten als Objekte im Cluster gespeichert sind. Um beim Ausfall eines MDS schnell auf einen anderen umschalten zu können, ohne, dass dieser erst umständlich die Metadaten in seinen Speicher laden muss, können Standby-MDS per Konfiguration vorgehalten werden, die einem aktiven MDS folgen und so im Notfall die entsprechenden Metadaten schon im Hauptspeicher geladen haben. Da Ceph hochskalierbar zu sein verspricht und auch mit großen Mengen Metadaten umgehen können muss, dürfen mehrere MDS aktiv sein, die jeweils einen anderen Metadatenteil des Dateisystems vorhalten. Nicht-aktive Standby-MDS stellen die Verfügbarkeit sicher.

Erasure-Coding - Ist die Speicherung ganzer Repliken von Objekten unerwünscht, weil dafür bspw. zu viel Speicherplatz beschrieben wird, kann das Erasure-Coding eingesetzt werden [9]. Es funktioniert praktisch wie ein RAID mit frei wählbarer Parität. Jedes Objekt wird in n Teile, Chunks genannt, gesplittet, wobei es k Data-Chunks und m Paritäts-Chunks, Coding-Chunks genannt, gibt. Die Zahl m wird auch Reliability-Level genannt. Als Formel ergibt sich:

$$n = k + m$$

Um nach einem Hardwareausfall Daten wiederherzustellen, sind immer k Chunks nötig. Dabei spielt es keine Rolle, ob es sich um Data-Chunks oder Coding-Chunks handelt. Sind mehr als k Chunks ausgefallen, ist eine Wiederherstellung nicht möglich. Folgendes Beispiel sei erläutert: Es wird eine Datei mit dem Textinhalt „Das ist ein Beispiel“ im Cluster gespeichert. Es sei ein Reliability-Level von 2 eingestellt. Dann wird die Datei bspw. in drei Teilobjekte mit dem Inhalt: „Das i“, „st ein B“ und „ispiel“ geteilt und mithilfe eines Algorithmus zwei zusätzliche Objekte (Coding-Chunks) generiert, die bspw. den unverständlichen Inhalt „ikaunsd“ und „aswefw“ haben können. Die fünf erzeugten Objekte werden nun auf ebensoviele OSDs verteilt, die möglichst in verschiedenen Ausfallzonen liegen. Fallen nun zwei betroffene OSDs aus, kann aus den verbliebenen drei Chunks, egal ob Data- oder Coding-Chunk, die ursprüngliche Datei wiederhergestellt werden. Erasure-Coding funktioniert damit im einfachsten Falle wie ein RAID 5 (mit einfacher Parität) oder wie ein RAID 6 (mit zweifacher Parität). Allerdings kann durch die beliebig einstellbare Anzahl an Coding-Chunks die Parität ebenso wahlfrei festgelegt werden.

Filestore - Ceph kennt zwei Speicherbackends bzw. Art und Weisen, wie es seine Dateobjekte auf den Ceph-Servern speichert: Filestore und Bluestore. Filestore ist das ältere und bewährte Backend und defaultmäßig installiert. Im Filestore gibt es, wie beschrieben,

OSDs und Journale, auf denen die Daten gespeichert werden. Auf einem OSD ist ein Linux-Dateisystem installiert und die Datenobjekte werden in Form von ganz normalen Dateien gespeichert.

Bluestore - Im Gegensatz dazu gibt es seit Version „jewel“ den Bluestore [10] (**B**lock + **N**ew**S**tore), der ohne Journal und Dateisystem auf den OSDs auskommt. Er stellt eine Weiterentwicklung des Filestores dar, da er Datenobjekte direkt auf das Block-Device (das eigentliche OSD) schreibt, ohne, dass ein Dateisystem darauf nötig ist[2]. Der Filestore hat nämlich den Nachteil, dass im Prinzip immer zwei Journale beschrieben werden müssen, das Ceph-Journal und dasjenige des Dateisystems auf dem OSD, da die verwendbaren Dateisysteme selbst ein Journal mit sich bringen. Dieser Overhead wird bei Bluestore vermieden.

Die Metadaten für das unformatierte Block-Device werden in einer RocksDB-Datenbank [11] gespeichert, die entweder auf dem selben Device oder einem separaten, evtl. schnelleren Laufwerk (bspw. einer SSD) gespeichert werden kann. Neben der Metadatenbank muss es noch ein WAL (write-ahead-log) geben, das ebenso auf dem OSD selbst, oder einer schnelleren SSD liegen kann. Im WAL werden alle Schreibvorgänge persistiert, aufgehoben und nach einer bestimmten Zeit gelöscht. Es dient dazu, bspw. nach einem Absturz die Metadatenbank wiederherstellen zu können. Im folgenden Schaubild sei dies für den Fall, dass Metadatenbank und WAL auf dem OSD liegen, dargestellt.

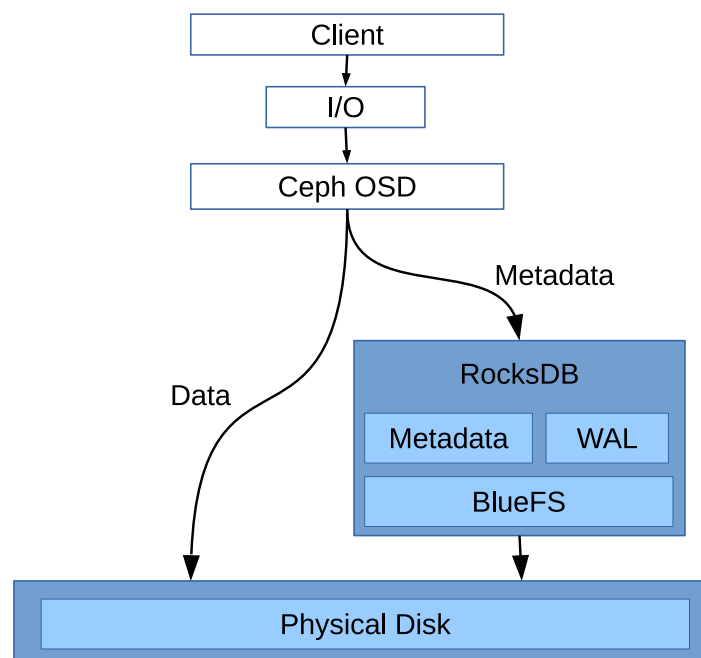


Abbildung 2.7: Aufbau eines OSD im Bluestore [2]

2.2 Speicherinterfaces

Ceph bietet drei verschiedene Arten von Speicherinterfaces an, mit deren Hilfe ein Client auf den Cluster zugreifen kann. RADOS Block-Device, RADOS Object-Gateway und ein verteiltes Dateisystem: CephFS.

2.2.1 RADOS Block-Device

Bei dieser Art von Speicher handelt es sich um unpartitionierten Blockspeicher [1, S. 53f], der in beliebiger, im Rahmen der Möglichkeiten des Clusters liegender, Größe einem Client bereitgestellt werden kann. Dieses Block-Device kann mit einem Dateisystem formatiert werden. Mithilfe des „mount“-Befehles wird dieses Dateisystem dann eingebunden. Die einzelnen Blöcke des Block-Devices liegen in Ceph-Objekten und werden gemäß den geltenden Replikationsregeln über den Cluster verteilt. Seit der Linux Version 2.6.39 ist der RADOS Block-Device-Treiber im Kernel integriert. Die Block-Devices bieten u.a. folgende Eigenschaften:

- Thin Provisioning
 - Das Image belegt zunächst keinen (oder nur sehr geringen) Speicherplatz. Seine tatsächliche Größe wächst erst, wenn Daten hineingeschrieben werden.
- copy-on-write
 - Kopien von Images benötigen zunächst (fast) keinen zusätzlichen Speicher. Erst wenn sich Daten in Kopie oder Original ändern, werden die Daten dereferenziert und kopiert.
- Snapshots
 - Durch die copy-on-write Fähigkeit ist das schnelle Erstellen von read-only Snapshots möglich. Aus diesen Abbildern können schreibfähige Klone erstellt werden, die zunächst vom Eltern-Image abhängig sind. Durch die „flatten“-Funktion ist es möglich, Klone in eigenständige Images zu verwandeln.

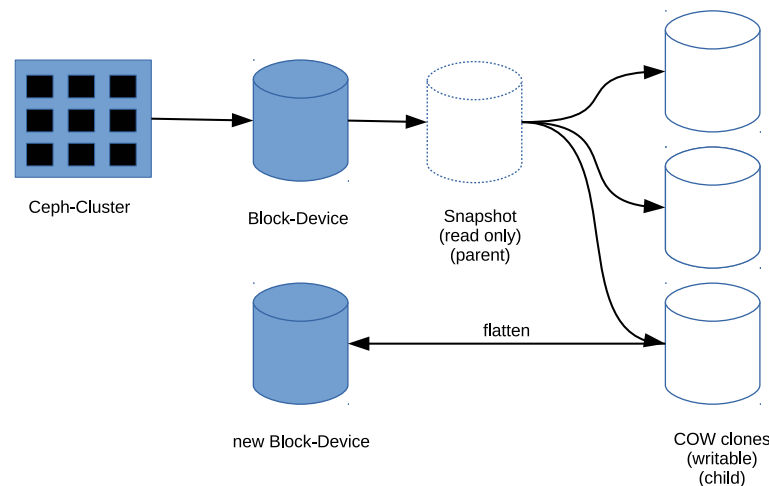


Abbildung 2.8: Snapshots und Klone [1, S. 114]

Aufgrund der genannten Eigenschaften ist der häufigste Anwendungsfall von RADOS Block-Devices die Bereitstellung von Speicher für die Images Virtueller Maschinen, wie es bspw. mit Ceph als Backend für Cinder in OpenStack geschieht. Durch die Möglichkeit, platzsparende Snapshots zu machen, lassen sich so schnell großen Zahlen von virtuellen Maschinen ausrollen.

2.2.2 RADOS Object-Gateway

Diese Form der Speicheranbindung [1, S. 54f] ermöglicht Zugriff auf Objekte über eine RESTful HTTP/S API. Die größeren REST Objekte werden von RADOS in kleinere Ceph-Objekte geteilt und wie die anderen Objekte auch über den Ceph-Cluster verteilt, wodurch sich die Lesegeschwindigkeit erhöhen soll. Da diese Form der Datenanbindung für den Auftraggeber dieser Studie nicht interessant ist, wird darauf bei den sich anschließenden Betrachtungen nicht weiter eingegangen werden.

Anwendung findet diese Form der Speicheranbindung bspw. bei Amazons S3 oder als Ersatz für Swift in OpenStack-Clustern, da beide eine RESTful API voraussetzen, die das RADOS Object-Gateway bereitstellt.

2.2.3 Ceph-Filesystem

Beim Ceph-Filesystem (CephFS) [8] handelt es sich um ein weitgehend Posix-kompatibles verteiltes Dateisystem, das auf verschiedene Weise in einen Client eingebunden werden

kann. Zum Betrieb eines CephFS ist mindestens ein MDS nötig. Die verschiedenen Anbindungsmöglichkeiten sind im folgenden Schaubild verdeutlicht.

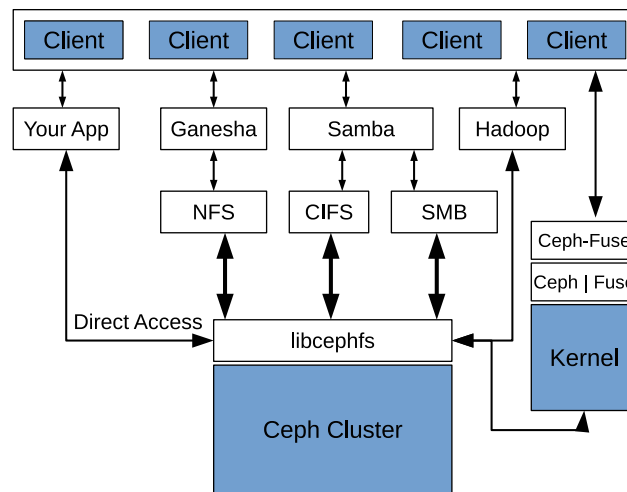


Abbildung 2.9: CephFS [1, S. 57]

An Unterschieden zu Posix seien zwei Merkmale genannt. Vier Weitere finden sich auf den Ceph-Doc-Seiten im Netz [12].

1. Wird eine Datei von mehreren Clients in den Page-Cache geladen und anschließend geändert, erscheint die Änderung nicht kohärent im Page-Cache aller betroffenen Clients.
2. Ceph erstellt ein Verzeichnis „snap“, in dem sich Informationen zu den möglichen Snapshots befinden. Obwohl das Verzeichnis versteckt und von „readdir“-Aufrufen ausgeschlossen ist, tritt eine Erscheinung ein, die nicht mit Posix kompatibel ist. Es kann kein gleichnamiges Verzeichnis erstellt werden. Abhilfe schafft der Parameter: „-o snapdirname=somethingelse“, mit dem man zur Zeit der Einbindung des Dateisystems wenigstens den Name des versteckten Verzeichnisses ändern kann.

Um das Dateisystem mithilfe kerneigener Module mounten zu können, benötigt ein Client auf Debian-basierten Systemen mindestens eines der beiden Pakete „ceph-fs-common“ oder „ceph-fuse“.

Die anderen Arten der Einbindung, etwa über NFS, CIFS und SMB, finden sich in der verwendeten Literatur [1, S. 57] wieder, werden in dieser Arbeit aber nicht eruiert. Sie sind theoretisch unter Zuhilfenahme eines Gateways möglich, werden aber in den offiziellen Ceph-Docs nicht weiter beschrieben.

2.3 Funktionsweise

Die Funktionsweise von Ceph soll anhand eines exemplarischen Schreibvorganges [1, S. 75f] erläutert werden. Dieser Ablauf ist für das RADOS Block-Device und das Ceph-Filesystem vergleichbar. Nur mit dem Unterschied, dass beim RADOS Block-Device die Datei vom lokalen Dateisystem erst noch in Blöcke aufgelöst wird, während beim Ceph-Filesystem die Dateien direkt in Objekte umgewandelt werden.

Auf dem **Client**:

1. Ein Client will eine Datei schreiben.
2. RADOS formt auf dem Client aus der Datei Objekte.
3. Der CRUSH-Algorithmus berechnet auf dem Client die ID der PG, in der die Objekte liegen sollen und damit auch die betreffenden OSDs, gemäß den Replikationseinstellungen.

Auf den **Servern**:

4. Die Objekte werden in das Journal des primären OSDs geschrieben.
5. Sind die Objekte in die Journale der sekundären OSDs geschrieben worden, senden diese ein Acknowledgement-Signal zum primären OSD.
6. Wenn alle Journale Vollzug gemeldet haben, sendet der primäre OSD dem Client ein Acknowledgement und der Speichervorgang ist für den Client beendet.
7. Die Objekte werden aus den Journalen in die OSDs geschrieben.
8. Der Speichervorgang ist beendet.

Der Umstand, dass der Client sein Acknowledgement erst erhält, wenn die Objekte in alle betreffenden Journale geschrieben sind, macht deutlich, dass deren Leistungsfähigkeit entscheidend für die Schreibgeschwindigkeit des Clusters ist [13].

2.3.1 Lage der Datenobjekte

Wie schon erwähnt worden ist, speichert Ceph seine logischen Objekte in physischen Dateien ab (Filestore-Prinzip, siehe Kapitel 2.1). Diese liegen in Dateisystemen auf den Servern, auf denen ein beliebiges Linux installiert ist. Zum besseren Verständnis sei hier ein Weg aufgezeigt, die Lage der Datenobjekte im Cluster zu bestimmen und die physischen Dateien zu finden [1, S. 76f].

Zu Demonstrationszwecken wird im Pool „test“ ein Ceph-Datenobjekt erzeugt, das den Inhalt einer beliebigen Datei (hier /etc/hosts) erhalten soll. Der Vorgang des Erzeugens eines derart kleinen Objektes geschieht rasend schnell und ist für menschliches Zeitempfinden nicht wahrnehmbar. Die Messung mit vor und nach dem Erstellen ausgeführten Zeitkommandos ergab 66 Millisekunden.

Listing 2.1: Ceph-Datenobjekt erzeugen

```
1 # erzeugt ein Objekt "object1" mit dem Inhalt der Datei "/etc/hosts" im Pool "test"
root@kokos:~ > rados -p test put object1 /etc/hosts
```

Dann lässt man sich anzeigen, in welcher PG und auf welchen OSDs das Objekt gespeichert wurde. Man erhält die PG-ID (1.bc) und die beteiligten OSDs (2,8,7), wobei 2 das primäre OSD ist.

Listing 2.2: Beteiligte OSDs ermitteln

```
# zeigt an, zu welchem Pool und welchen PGs das Objekt "object1" gehört
root@kokos:~ > ceph osd map test object1
3 osdmap e49 pool 'test' (1) object 'object1' -> pg 1.bac5debc (1.bc) -> up ([2,8,7], p2
  ) acting ([2,8,7], p2)
```

Jetzt bestimmt man die Lage der OSDs im Cluster. OSD.2 ist in koala, OSD.8 in kokos und OSD.7 im Rechner cobalt verbaut.

Listing 2.3: Lage der OSDs ermitteln

```
# gibt die Cluster-Map aus
2 root@kokos:~ > ceph osd tree
ID WEIGHT TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 5.89322 root default
-2 2.03107  host kokos
  0 0.44969      osd.0 up          1.00000          1.00000
  7 1 0.67699      osd.1 up          1.00000          1.00000
  8 0.90439      osd.8 up          1.00000          1.00000
-3 1.93108  host koala
  2 0.44969      osd.2 up          1.00000          1.00000
  3 0.57700      osd.3 up          1.00000          1.00000
12 4 0.90439      osd.4 up          1.00000          1.00000
-4 1.93108  host cobalt
  5 0.44969      osd.5 up          1.00000          1.00000
  6 0.57700      osd.6 up          1.00000          1.00000
  7 0.90439      osd.7 up          1.00000          1.00000
```

Zuletzt führt man eine Reihe von Kommandos auf einem der ermittelten Rechner aus (bspw. koala), die letztlich in ein Verzeichnis führen, in dem die Objekte der PG 1.bc

gespeichert sind. Am Ende erhält man die gesuchte Datei, die exakt den Inhalt der eingangs übergebenen Datei `/etc/hosts` aufweist.

Listing 2.4: Ceph-Datenobjekt finden

```
# mit koala verbinden
root@kokos:~ > ssh koala

4 # den Mountpoint des OSDs "osd.2" herausfinden
root@koala:~ > df -h | grep ceph-2
/dev/sdb1 461G 36M 461G 1% /var/lib/ceph/osd/ceph-2

# in das Datenverzeichnis des OSDs wechseln
9 root@koala:~ > cd /var/lib/ceph/osd/ceph-2/current

# alle PGs anzeigen lassen
root@koala:/var/lib/ceph/osd/ceph-2/current > ls -l | grep 1.bc
drwxr-xr-x 2 ceph ceph 65 Jun 2 10:43 "1.bc_head"/
14 drwxr-xr-x 2 ceph ceph 58 Jun 1 20:57 "1.bc_TEMP"/

# in das PG-Verzeichnis wechseln
root@koala:/var/lib/ceph/osd/ceph-2/current > cd 1.bc_head/

19 # alle Objekte der PG auflisten
root@koala:/var/lib/ceph/osd/ceph-2/current/1.bc_head > ls -l
total 4
-rw-r--r-- 1 ceph ceph 0 May 30 16:29 "__head_000000BC__1"
-rw-r--r-- 1 ceph ceph 169 Jun 2 10:43 "object1__head_BAC5DEBC__1"

24 # Inhalt des Objektes "object1" anzeigen
root@koala:/var/lib/ceph/osd/ceph-2/current/1.bc_head > cat object1__head_BAC5DEBC__1
127.0.0.1 localhost

29 # The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Kapitel 3

Administration eines Ceph-Clusters

Um einen Ceph-Cluster benutzen zu können, sind einige essentielle Kommandos und Abläufe nötig. Die Arbeit mit Pools sowie die Einbindung von Block-Devices und dem Ceph-Filesystem sind so wichtig, dass sie hier in aller Kürze dargestellt werden. Weitere Informationen über Befehle auf der Kommandozeile sind in den Ceph-Docs [8] sowie in der gedruckten Literatur [1], [7], [9] zu finden.

3.1 Pools und Placement-Groups

Der dem Ceph-Cluster in Form von OSDs zur Verfügung stehende Speicher wird logisch in Pools unterteilt, die erstellt und ggfs. wieder gelöscht werden können. Diese Pools wiederum sind in Placement-Groups unterteilt, in welchen sich die einzelnen Datenobjekte befinden (siehe Abbildung 2.5 auf Seite 19). Wird ein Pool erstellt, muss Ceph mitgeteilt werden, aus wie vielen Placement-Groups er bestehen soll. Wächst der Cluster mit der Zeit an, muss die Anzahl der PGs erhöht werden. Diese Anzahl ist in drei Situationen von Bedeutung:

1. Bei Wiederherstellungsoperationen (Recovery) [14]. Die PG-Anzahl muss an der Größe des Clusters ausgerichtet sein. Untenstehende, nur der Demonstration des Prinzips dienende, Tabelle 3.1 zeigt eine fiktive (nicht gemessene) Zeit, die nötig sei, um eine Wiederherstellung abzuschließen. Ebenso wird gezeigt, wie sich die Anzahl der an der Wiederherstellung beteiligten OSDs in Abhängigkeit vom Replikationslevel und der Anzahl an OSDs in einem Cluster bei gleichbleibender PG-Anzahl berechnet. Bei Ausfall eines OSDs versucht Ceph, die verlorenen PGs wiederherzustellen und gleichmäßig über den Cluster zu verteilen. Die dafür nötige Recovery-Zeit sinkt bis zu einem gewissen Grad durch das Hinzufügen von OSDs zum Cluster. Überschreitet der Cluster aber eine bestimmte Anzahl an OSDs und die PG-Anzahl wird nicht erhöht, wird die Anzahl an PGs pro OSD so gering, dass bei Ausfall eines OSDs nur noch wenige verbliebene OSDs an der Wiederherstellung beteiligt

sind und begrenzende Faktoren wie Geschwindigkeit der Festplatten und Netzwerkdurchsatz die Recovery-Zeit beschränken. Die Anzahl involvierter OSDs berechnet sich wie folgt (am Beispiel der letzten Zeile): Pro OSD gibt es nur acht PGs. Diese fallen aus, da ein OSD ausfällt. Sie müssen auf acht anderen OSDs verteilt werden und erhalten von maximal zwei mal acht OSDs Kopien, da der Replikationsfaktor bei drei liegt. Daraus ergibt sich, dass maximal 24 OSDs beteiligt sind. Wäre diese Zahl, wie in den anderen Zeilen, größer als die Anzahl noch verfügbarer OSDs, müssten alle verbliebenen OSDs an der Wiederherstellung teilnehmen, was Geschwindigkeitsvorteile mit sich bringt.

Repliken	OSDs	PGs	PGs pro OSD	involvierte OSDs	Recovery-Zeit
3	10	512	$\frac{512 \cdot 3}{10} \approx 150$	$150 \cdot 3 = 450 \rightarrow 9$	10 min
3	20	512	$\frac{512 \cdot 3}{20} \approx 75$	$75 \cdot 3 = 225 \rightarrow 19$	5 min
3	40	512	$\frac{512 \cdot 3}{40} \approx 38$	$38 \cdot 3 = 114 \rightarrow 39$	2,5 min
3	200	512	$\frac{512 \cdot 3}{200} \approx 8$	$8 \cdot 3 = 24 \rightarrow 24$	4,5 min

Tabelle 3.1: Recovery bei Ausfall eines OSDs

2. Bei der gleichmäßigen Datenverteilung im Cluster [14]. CRUSH teilt Datenobjekte gleichmäßig den PGs zu und verteilt diese PGs ebenso gleichmäßig über den Cluster. Eine genügend große Anzahl an PGs trägt damit zur gleichmäßigen Auslastung der OSDs bei. Dies sei an einem einfachen Beispiel verdeutlicht. Ein Cluster habe 10 OSDs und nur eine PG bei einem Replikationslevel von drei. CRUSH bliebe nichts anderes übrig, als alle Datenobjekte in diese eine PG zu stecken und sie auf drei OSDs zu verteilen. Es wären also nur drei OSDs von 10 in Gebrauch. Eine Erhöhung der PG-Anzahl würde hier eine gleichmäßigere Auslastung der OSDs bewirken.

3. Bei der Auslastung der Server [14]. Für jede PG reservieren OSD-Daemons und Monitore permanent Arbeitsspeicher, Netzwerkkapazität und Prozessorleistung. Durch eine genügend große Anzahl an PGs wird dieser Overhead gleichmäßig auf alle Server verteilt.

Nachdem die Bedeutung der PGs erläutert wurde, kann man nun daran gehen, ihre ideale Anzahl zu berechnen. Ceph bietet einen Rechner im Web an, der dem Administrator dabei hilft [15]. Dem Web-Tool liegt folgende Formel zugrunde:

$$\frac{(\text{Target PGs per OSD}) \cdot (\text{Total Number of OSDs}) \cdot (\% \text{ Data})}{\text{Size}} = \text{Anzahl der PGs}$$

Faktor	Beschreibung
Target PGs per OSD	Anzahl der PGs, die einem OSD zugeordnet sein sollen. Der Wert sollte zwischen 100 und 300 liegen, je nachdem, ob der Cluster demnächst erweitert werden soll. (100 ist das eigentliche Ziel. Alles darüber nur bei anstehender Erweiterung.)
Total Number of OSDs	Anzahl der OSDs, die im Ceph-Cluster eingebunden sind.
% Data	Bei mehreren Pools gibt dieser Wert an, wie groß ein Pool in Relation zum gesamten Cluster ist.
Size	Replikationslevel

Tabelle 3.2: Berechnung der Anzahl an Placement-Groups

Die Formel berechnet die PG-Anzahl für einen Pool. Soll es mehrere Pools in einem Cluster geben, muss über den Faktor „%Data“ jeder einzelne Pool je nach Menge der in ihm enthaltenden Daten gewichtet werden. Laut den Empfehlungen der offiziellen Ceph-Dokumentation soll das Ergebnis zur nächstliegenden Potenz von zwei gerundet werden, um „CRUSH die gleichmäßige Verteilung von Objekten in Placement-Groups“ [16] zu ermöglichen.

Konfiguration

Im weiteren Verlauf soll der Einfachheit halber angenommen werden, dass es nur einen Pool gibt, 100 PGs pro OSD erstellt werden sollen und es 9 OSDs, bei einer Replikationszahl von drei gibt. Daraus ergibt sich nach der oben genannten Formel eine PG-Anzahl von 300 für den Pool. Diese wird zur nächsten Potenz von zwei abgerundet und beträgt damit 256.

Listing 3.1: Erstellung und Löschung eines Pools

```
# legt einen Pool "test" mit 256 PGs an
root@kokos:~ > ceph osd pool create test 256 256

# löscht denselben Pool
root@kokos:~ > ceph osd pool delete test test --yes-i-really-really-mean-it
```

Damit besteht nun das Grundgerüst, um mit der Nutzung des Ceph-Clusters fortzufahren und bspw. Images für die RADOS Block-Devices erstellen oder Ceph-Filesysteme anlegen zu können.

3.2 RADOS Block-Devices

Um ein RADOS Block-Device auf einem Client nutzen zu können, muss auf diesem Ceph zunächst ordnungsgemäß installiert und eingerichtet werden. Im Folgenden wird Ceph auf

dem Client „berkelium“ installiert, der entsprechende Authentifikationskeyring und die Cluster-Konfigurationsdatei „ceph.conf“ auf ihn kopiert und schließlich der Keyring auf dem Client „berkelium“ lesbar gemacht.

Listing 3.2: Vorbereiten des Clients

```
# Ceph installieren (vom Admin-Node aus)
root@kokos:~ > ceph-deploy install berkelium

# ceph.conf und Keyring kopieren auf den Client kopieren (vom Admin-Node aus)
5 root@kokos:~ > ceph-deploy admin berkelium

# Keyring lesbar machen (auf dem Client)
root@berkelium:~ > sudo chmod +r /etc/ceph/ceph.client.admin.keyring
```

Die nun folgenden Operationen binden ein Block-Device als Laufwerk in einen Client ein. Zu Beginn wird ein Block-Device auf dem Ceph-Cluster erstellt (die Verbindungsdaten bezieht der Client aus der kopierten „ceph.conf“), dann gemapped, mit einem Dateisystem formatiert und als gemountetes Laufwerk in das System eingebunden. Folgende Befehle führen diesen Prozess aus:

Listing 3.3: Einbinden eines Block-Devices

```
# erstellt Block-Device namens "test_device" im Pool "test" der Größe 40 GB
2 root@berkelium:~ > rbd create test_device -p test --size 40G

# macht das Block-Device unter "/proc/partitions" sichtbar
root@berkelium:~ > rbd map test_device -p test --name client.admin

7 # erstellt ein Dateisystem auf dem Block-Device
root@berkelium:~ > mkfs.ext4 -m0 /dev/rbd0

# mountet das Block-Device nach "/mnt"
root@berkelium:~ > mount /dev/rbd0 /mnt
```

Sollte man das das eingebundene Block-Device wieder löschen wollen, geht man umgekehrt vor:

Listing 3.4: Löschen eines Block-Devices

```
# Unmounten des Block-Devices
root@berkelium:~ > umount /mnt

4 # Block-Device aus der Liste der Block-Devices des Clients entfernen
root@berkelium:~ > rbd unmap test_device -p test --name client.admin

# Block-Device in Ceph löschen
root@berkelium:~ > rbd remove test_device -p test
```

3.3 Ceph-Filesystem

Ceph bringt ein eigenes verteiltes Dateisystem mit, das auf Clients gemountet werden kann.

Um ein Ceph-Filesystem auf dem Ceph-Cluster zu betreiben, ist mindestens ein Metadata-Server nötig. Aus Gründen der Ausfallsicherheit sollten aber wenigstens zwei installiert sein, wobei einer führend ist und der andere nur im Bedarfsfall einspringt.

Will man ein Ceph-Filesystem auf dem Cluster einrichten, müssen zunächst zwei Pools angelegt werden. Einer für die eigentlichen Daten und einer für Metadaten, um bspw. Dateisystembefehle wie „ls“ oder „find“ ausführen zu können. Nachdem die beiden Pools angelegt sind, wird schließlich das Dateisystem selbst angelegt. Zu beachten ist, dass „kokos“ ein Ceph-Server ist, die Dateisystemerstellungsoptionen also von ihm aus gehen.

Listing 3.5: Erstellen eines Ceph-Filesystems

```
# Pool "cephfs_data" für Daten und "cephfs_metadata" für Metadaten erstellen
2 root@kokos:~ > ceph osd pool create cephfs_data 256
root@kokos:~ > ceph osd pool create cephfs_metadata 256

# Dateisystem "testFS" erstellen
root@kokos:~ > ceph fs new testFS cephfs_metadata cephfs_data
```

Der Client muss sich am Cluster authentifizieren. Dazu wird je nach Nutzer etwa aus dem Keyring „ceph.client.admin.keyring“ das Passwort extrahiert und in einer Datei bspw. „admin.secret“ gespeichert. Diese Datei kann man mit entsprechenden Leserechten ausstatten und später beim Mount übergeben. Der Vorteil gegenüber der Klartext-Passwortübergabe liegt darin, dass das Passwort nicht in der History auftaucht und man die Datei vor unerlaubtem Lesen schützen kann. Um das Dateisystem mounten zu können, muss noch das Paket „ceph-fs-common“ installiert werden. Letztendlich wird der Befehl „mount“ ausgeführt und das Dateisystem mithilfe eines Treibers im Kernel eingebunden. Wiederum ist „kokos“ ein Ceph-Server und „berkelium“ ein Client.

Listing 3.6: Einbinden von CephFS mit Kernelmodul

```

# vom Admin-node aus Ceph auf dem Client "berkelium" installieren, wenn dies noch
  nicht schon geschehen ist
root@kokos:~ > ceph-deploy install berkelium

4 # Extrahieren des Passwortes und Speichern in einer zu schützenden Datei
root@kokos:~ > cat ceph.client.admin.keyring | grep -A 1 "client.admin" | grep "key ="
  " | awk '{ print $3 }' > admin.secret

# Schützen der Datei gegen unerlaubtes Lesen
root@kokos:~ > chmod 700 admin.secret

9 # Kopieren der geheimen Datei auf den Client (in ein beliebiges Verzeichnis)
root@kokos:~ > scp admin.secret berkelium:/root/my-cluster

# Installation des nötigen Pakets auf dem Client "berkelium"
14 root@berkelium:~ > apt-get install -y ceph-fs-common

# Einbinden unter /mnt
root@berkelium:~ > mount -t ceph {ip-address-of-monitor}:6789:/ /mnt -o name=admin,
  secretfile=admin.secret

```

Um mittels „mount“ vorzugehen, muss der Kernel mindestens die Version 2.6.34 haben, da der Ceph-Dateisystem-Treiber erst ab dieser Version im Kernel integriert ist. Benutzt man eine ältere Kernelversion, kann man das Dateisystem mittels „fuse“ einbinden. Dazu muss der entsprechende Keyring auf den Client kopiert und beim Mount übergeben werden. Außerdem wird zusätzlich das Paket „ceph-fuse“ benötigt.

Listing 3.7: Einbinden von CephFS mit Fuse

```

# Kopieren des Keyrings auf den Client
root@kokos:~ > scp ceph.client.admin.keyring berkelium:/root/my-cluster

3 # Installation des nötigen Pakets
root@berkelium:~ > apt-get install -y ceph-fuse

# Einbinden unter "/mnt"
8 root@berkelium:~ > ceph-fuse -k ceph.client.admin.keyring -m {ip-address-of-monitor}
  }:6789 /mnt

```

Bei beiden Varianten steht im Anschluss das Dateisystem unter „/mnt“ zur Verfügung und kann getestet werden.

3.4 Schutz vor unberechtigtem Zugriff

Es ist ratsam, beim Betrieb eines produktiven Ceph-Clusters Maßnahmen zum Schutz vor unberechtigtem Zugriff zu ergreifen. Dies geschieht durch das User-Management von

Ceph [17]. Im Grundsatz funktioniert es so, dass man einen Nutzer und mit ihm einen Schlüssel erstellt, der dann bestimmte Rechte auf dem Cluster hat. Einen Nutzer, der bspw. Leserechte für Monitore hat, sich also mit dem Cluster verbinden kann und über Lese- und Schreibrechte bspw. für den Pool „liverpool“ verfügt, erstellt man folgendermaßen:

Listing 3.8: Erstellen eines Nutzers

```
# erstellt einen Nutzer mit Leserecht für einen Monitor und Lese- und Schreibrecht auf
    dem Pool "liverpool"; gibt eine "key"-Datei zurück
2 root@kokos:~ > ceph auth get-or-create-key client.joerg mon 'allow r' osd 'allow rw
    pool=liverpool' -o joerg.key
```

Das vorgestellte Kommando gibt eine Schlüsseldatei zurück, die nun auf den Client kopiert werden muss, von dem aus Zugriff auf ein Speicher-Interface des Clusters erfolgen soll. Der Schlüssel selbst wird bei den entsprechenden Kommandos zur Erzeugung und zum Einbinden bspw. eines Block-Devices auf dem Client als Parameter übergeben. Wird der Schlüssel entwendet, ist der Schaden begrenzt, insofern die passenden Rechte für ihn gesetzt worden sind. Das bedeutet, ein Angreifer bzw. Erbeuter des Schlüssels kann nur den Schaden anrichten, der dem Schlüssel erlaubt ist. Wird allerdings ein Admin-Schlüssel auf einen Client kopiert und dort entwendet, kann der gesamte Ceph-Cluster zerstört werden.

Die Rechte eines Nutzers können geändert werden, je nachdem, wie viel ihm auf dem Cluster erlaubt sein soll. Im Folgenden seien einige Beispiele genannt (Weitere Rechte findet man in den Ceph-Docs [17].):

Listing 3.9: Veränderung der Rechte eines Nutzers

```
# verändert Rechte des Nutzers joerg; setzt zusätzlich das Schreibrecht für Objekte
    und das Ausführrecht im Pool "liverpool"
root@kokos:~ > ceph auth caps client.joerg mon 'allow rw' osd 'allow rwx pool=
    liverpool'
3
# verändert Rechte des Nutzers joerg; gibt ihm volle Adminrechte
root@kokos:~ > ceph auth caps client.joerg mon 'allow *' osd 'allow *'

# verändert Rechte des Nutzers joerg; gibt ihm Lese- und Schreibrechte für das Ceph-
    Filesystem
8 root@kokos:~ > ceph auth caps client.joerg mds 'allow'

# verändert Rechte des Nutzers joerg; erlaubt ihm, den Pfad /irgendein/Pfad mit Lese-
    und Schreibrechten zu mounten
root@kokos:~ > ceph fs authorize NameEinesFilesystems client.joerg /irgendein/Pfad rw
```

Mithilfe der vorgestellten Methoden kann der Cluster vor Schaden durch Verlust von Schlüsseln bewahrt werden. Es lassen sich Pools im ganzen Ceph-Cluster und Pfade im

Ceph-Filesystem schützen. Im weiteren Verlauf dieser Arbeit wird trotzdem mit dem Admin-Key gearbeitet werden, weil das auf einem Test-Cluster kein Problem darstellt, da keine wertvollen Daten auf ihm liegen. Im Produktivbetrieb sollten dagegen nutzerspezifische Schlüssel erzeugt und verwendet werden.

Kapitel 4

Monitoring-Tools

Um die Server des Ceph-Clusters während der Messungen zu überwachen und begrenzende Faktoren wie CPU, Netzwerk und Hauptspeicher im Blick zu behalten, bedarf es einer entsprechenden Software-Lösung, die diese Werte protokolliert und ansprechende Grafiken erzeugen kann. Zu diesem Zweck wurden die Programme `collectd` und `graphite` gewählt. Während `collectd` die Messdaten sammelt und verschickt, stellt sie `graphite` live auf einem Webinterface dar.

4.1 `collectd`

`Collectd` [18] ist ein Daemon, der, lokal auf einem Rechner ausgeführt, Messdaten erhebt und mit zahlreichen Plugins erweiterbar ist, die u.a. CPU-Auslastung, Hauptspeicher und verschiedene Netzwerkschnittstellen überwachen können. `Collectd` wurde gegenüber dem Paket „`sysstat`“ der Vorzug gegeben, da es über Plugins erweiterbar ist und einfach in `graphite` integriert werden kann. Im Folgenden sei die Inbetriebnahme von `collectd` mithilfe eines eigens erstellten Scriptes (siehe Anhang Listing A.1 auf Seite 121) dargestellt, mit dem sich Plugins bequem laden und entladen lassen. Es installiert `collectd` und lädt das Plugin „`Processes`“, das für die Überwachung der OSD-Daemons sorgt. Dabei muss darauf geachtet werden, dass sich eine Datei „`collectd.conf`“ (siehe Anhang Listing A.2 auf Seite 125) im selben Verzeichnis wie das Script befindet, da sie als Basiskonfiguration dient. Nachdem `collectd` auf jedem zu überwachenden Rechner installiert ist, schickt es mithilfe des `graphite`-Plugins seine Messdaten zum eingerichteten `graphite`-Server auf Port 2003.

Listing 4.1: Installation von `collectd`

```
# Nutze Script zur Erstellung von Konfigurationsdateien
root@kokos:~ > ./go_collectd.sh -iepsl processes
```

4.2 graphite

Graphite [19] ist ein Programm, mit dem man beliebige Messdatenreihen live auf einer Weboberfläche in Form von Diagrammen anzeigen lassen kann. Es erhebt selbst keine Daten, speichert sie aber und zeigt sie an. Die Speicherung erfolgt in einer Whisper-Datenbank [20], deren markanteste Eigenschaft es ist, dass sie nicht über ein bestimmtes Maß hinaus wächst. Zum Zeitpunkt ihrer Erstellung wird der Datenbank mitgeteilt, wie viele Minuten, Stunden oder Tage sie Daten aufzeichnen soll. Wird der Zeitraum überschritten, werden die ältesten Daten mit den neuesten überschrieben. Eine Verdichtung in Form von Kompression oder Mittelwertberechnung, wie bei üblichen „RRD“ (round-robin-databases), erfolgt dabei nicht. Will man die Grafiken aus graphite heraus speichern, kann man dies einfach mithilfe der „Bild speichern unter“-Funktion des Browsers erledigen.

Im Folgenden sei der Weg dargelegt, graphite auf einem Ubuntu 16.04 zu installieren.

Listing 4.2: Installation von graphite

```
# Installation der nötigen Pakete
root@sellerie:~ > apt-get update; apt-get dist-upgrade
3 root@sellerie:~ > apt-get install build-essential graphite-web graphite-carbon python-
    dev libapache2-mod-wsgi libpq-dev python-psycopg2 apache2

# Das Processing-Backend Carbon wird benötigt. Starte es beim Boot.
root@sellerie:~ > sed -i 's/CARBON_CACHE.*/CARBON_CACHE_ENABLED=True/' /etc/default/
    graphite-carbon

8 root@sellerie:~ > service carbon-cache start
root@sellerie:~ > service carbon-cache status

# Anpassung der Zeitzone
root@sellerie:~ > sed -i 's/TIME_ZONE.*/TIME_ZONE = "Europe\Berlin"/' /opt/graphite/
    webapp/graphite/local_settings.py
13

# Deployment mit Apache2
root@sellerie:~ > a2dissite 000-default.conf
root@sellerie:~ > service apache2 reload
root@sellerie:~ > cp /usr/share/graphite-web/apache2-graphite.conf /etc/apache2/sites-
    available/
18 root@sellerie:~ > a2ensite apache2-graphite.conf
root@sellerie:~ > service apache2 reload
root@sellerie:~ > service apache2 status

# Erstellen einer Nutzerdatenbank für graphite
23 root@sellerie:~ > chown _graphite:_graphite /var/lib/graphite/graphite.db
root@sellerie:~ > graphite-manage syncdb
root@sellerie:~ > chmod 664 /usr/share/graphite-web/graphite.wsgi
```

Das Frontend von graphite ist nun unter der Adresse „IP-DES-HOSTS:80“ in einem Webbrowser erreichbar. Anpassungen des Sockets, bspw. eine andere IP oder ein anderer Port,

können in der Datei „`/etc/apache2/sites-available/apache2-graphite.conf`“ vorgenommen werden, wenn das Frontend auf einem anderen Port laufen soll.

Sind graphite und collectd korrekt konfiguriert, kann daran gegangen werden, ein Dashboard zu erstellen, welches aus den gewünschten Graphen besteht und live Messdaten anzeigt. Abbildung 4.1 zeigt das arithmetische Mittel der CPU-Auslastung aller CPUs, den Netzwerkdurchsatz und die Arbeitsspeicherbelegung der fünf Ceph-Server `osceph2`, `osceph3`, `osceph4`, `osceph5` und `osceph6` an. Die Daten werden mit einer zeitlichen Auflösung von einer Sekunde erhoben und live angezeigt. Wird ein Test ausgeführt, können so einfach Engstellen im System ausgemacht werden, die einer schnelleren Schreib- und Lese-rate evtl. im Wege stehen. Die Grafiken müssen dabei nicht unbedingt live am Bildschirm verfolgt werden. Man kann zu jedem beliebigen Zeitraum in der Vergangenheit zurückkehren und sich die Messwerte dieses Zeitraumes anzeigen lassen. Alles was man dafür benötigt, sind Anfangs- und Endzeitpunkt der Messung.

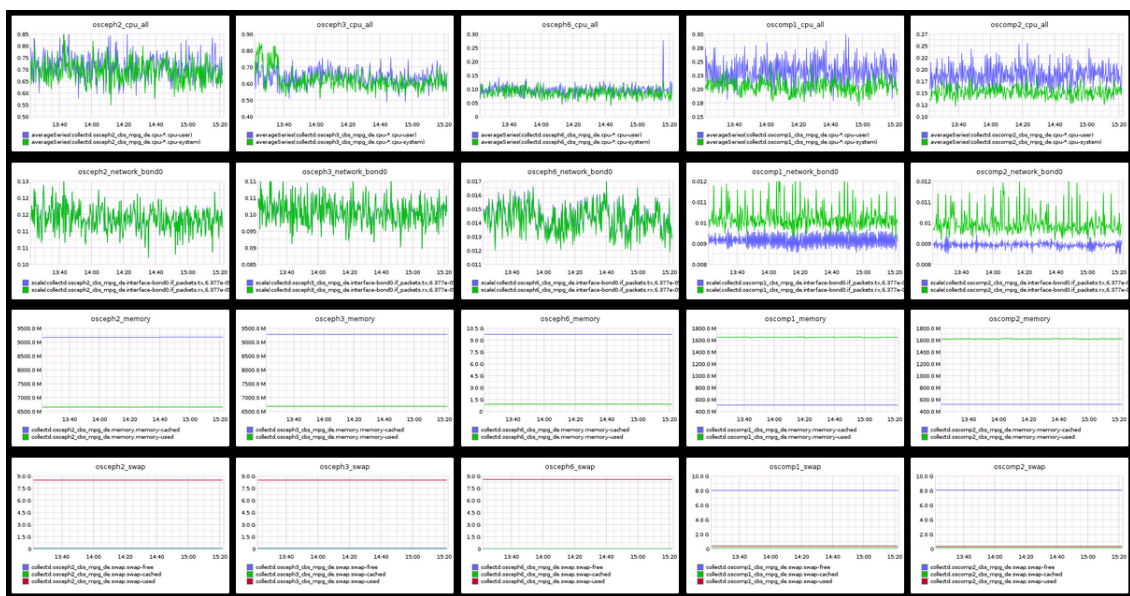


Abbildung 4.1: graphite Dashboard zur Überwachung limitierender Faktoren. Grafiken eines Ceph-Servers stehen untereinander.

4.3 ksysguard

Ein weiteres Tool zum Überwachen der Ceph-Server ist `kysysguard`. Es sei hier nur am Rande erwähnt, da es nicht zur Aufzeichnung der Monitoringdaten, sondern nur zur Liveüberwachung eingesetzt wurde. Es lässt sich einfach per Debian-Paket installieren und die Konfiguration ist selbsterklärend. Es funktioniert so, dass auf jedem zu überwachenden Server ein Daemon mit dem Debian-Paket „`kysysguardd`“ installiert wird und auf dem zur Anzeige dienenden Rechner das Frontend mit dem Paket „`kysysguard`“. Das Frontend ist in

QT geschrieben und sehr performant. Alles was man konfigurieren muss, ist die IP der zu überwachenden Server. Die Oberfläche zum Überwachen kann man sich dann einfach per Maussteuerung mit wenigen Klicks erstellen. Der Vorteil gegenüber graphite liegt darin, dass die Anzeige wesentlich schneller funktioniert als das Anzeigen der Grafiken im Browser. Dort gibt es bei graphite das Problem, dass der Bildaufbau teilweise zu lange dauert und der Eindruck des „Ruckelns“ entsteht.



Abbildung 4.2: ksysguard zur Überwachung limitierender Faktoren; Grafiken eines Ceph-Servers stehen untereinander.

Kapitel 5

Die Testsysteme

Die Tests des Ceph-Systems werden auf zwei Clustern durchgeführt. Einem relativ schwachen Commodity-Hardware-Cluster, auf dem das Versprechen Ceph's überprüft werden soll, auf derartiger Technik laufen zu können und einem Verbund aus professioneller Server-Hardware, der im weiteren Verlauf verschiedenen Optimierungen unterworfen werden wird.

5.1 Commodity-Hardware-Cluster

5.1.1 Hardware

In nachfolgender Tabelle ist die Hardwarekonfiguration des Commodity-Testsystems aufgelistet. Es besteht insgesamt aus 8 Rechnern gleicher Hardware (bis auf die Bestückung mit Festplatten). Die Namen der Rechner im Verbund lauten:

- kokos, koala, cobalt, thulium, thorium (Ceph-Server)
- berkelium, gadolinium (Ceph-Client)
- sellerie (Monitoring-Server)

Es handelt sich um Rechner, deren Hardware mit Bedacht schon recht alt und relativ schwach ist. Es sind einfache SATA II und III Festplatten und durchweg Intel Quadcores aus dem Jahre 2007 verbaut. Der RAM liegt bei 8 GB DDR2 und die Netzwerkschnittstelle leistet überall 1 GBit/s. Die nachfolgende Tabelle listet die Details auf:

Komponente	technische Beschreibung
CPU	Intel Core 2 Quad CPU Q6600, 4-Core, 2,40 GHz
RAM	8 GB DDR2
HDD, kokos	(System) 1 TB Hitachi Deskstar 7K1000.C (OSD) 500 GB Western Digital Caviar Blue (OSD) 750 GB Western Digital Caviar Blue (OSD) 1 TB Seagate SV35 (= 2,25 TB für Ceph)
HDD, koala	(System) 80 GB SAMSUNG HD080HJ/P (OSD) 500 GB Western Digital Caviar Blue (OSD) 640 GB Western Digital Caviar Black (OSD) 1 TB Seagate SV35 (= 2,14 TB für Ceph)
HDD, cobalt	(System) 750 GB Western Digital Caviar Green (OSD) 500 GB Western Digital Caviar Blue (OSD) 640 GB Western Digital Caviar Black (OSD) 1 TB Seagate SV35 (= 2,14 TB für Ceph)
HDD, thulium	(System) 1 TB Hitachi Deskstar 7K1000.C (OSD) 500 GB SAMSUNG SpinPoint T166 (OSD) 500 GB Western Digital Caviar Blue (OSD) 1 TB Seagate SV35 (= 2 TB für Ceph)
HDD, thorium	(System) 500 GB Western Digital Caviar Blue (OSD) 500 GB Western Digital Caviar Blue (OSD) 500 GB Western Digital Caviar Black (OSD) 1 TB Seagate SV35 (= 2 TB für Ceph)
HDD, berkelium	(System) 1 TB Seagate SV35
HDD, gadolinium	(System) 1 TB Seagate SV35
HDD, sellerie	(System) 750 GB Seagate Barracuda ES
Netzwerk	1 GBit Intel Corporation 82574L Gigabit Network Connection

Tabelle 5.1: Technische Daten des Commodity-Hardware-Clusters

5.2 Profi-Hardware-Cluster

Im folgenden die Hardwarekonfiguration des Profi-Hardware-Clusters. Er besteht aus 10 Servern. Die fünf Ceph-Server sind mit gleicher Hardware ausgestattet, die im Wesentlichen aus vielen Festplatten besteht. Die Clients und der Monitoring-Server sind ebenso mit gleicher Hardware versehen, die in diesem Falle mehr an Prozessorleistung und Arbeitsspeicher orientiert ist.

- osceph2, osceph3, osceph4, osceph5, osceph6 (Ceph-Server)
- oscomp1, oscomp2, oscomp3, oscomp4 (Ceph-Clients)

- oscomp5 (Monitoring-Server)

5.2.1 Hardware der Ceph-Server

Der Cluster beinhaltet fünf Ceph-Server, auf denen die OSDs gemountet sind. Diese Server sind darauf optimiert, Storage zur Verfügung zu stellen. Arbeitsspeicher und CPU sind von sekundärer Bedeutung. Dennoch empfiehlt die Literatur 1 GHz CPU-Taktfrequenz und 2 GB RAM pro OSD. Da die Server über 12 OSDs und 32 GB RAM verfügen, wird diese Vorgabe eingehalten. Des weiteren verfügen die Ceph-Server über je zwei SSDs, auf denen später die Journale installiert werden können, um Performancegewinne zu erzielen. Die Netzwerkanbindung erfolgt über je zwei 10 GBit Interfaces.

Komponente	technische Beschreibung
CPU	Intel Xeon Processor E5-1650 v2, 6-Core (12-Core mit HT), 3,5-3,9 GHz
RAM	32 GB DDR3
HDD	(System) 2 x 2 TB Hitachi/HGST Ultrastar 7K4000 in Software RAID 1 (OSD-Journale) 2 x 240 GB SAMSUNG MZ7WD240HCFV-00003 (OSDs) 12 x 4 TB Hitachi/HGST Ultrastar 7K4000 (= 48 TB für Ceph)
Netzwerk	2 x 10 GBit Intel Ethernet X520-SR2 (Glasfaser)

Tabelle 5.2: Technische Daten der Ceph-Server des Profi-Hardware-Clusters

5.2.2 Hardware der übrigen Server

Um die Ceph-Funktionalitäten testen zu können, werden noch Server gebraucht, um Last zu erzeugen (Clients) und Monitoring-Dienste zu leisten. Dazu wurden fünf weitere Server benutzt, die vor allem über viele CPUs (20 physische) und einen großen Arbeitsspeicher (256 GB) verfügen. Sie sind, wie auch die Ceph-Server, mit je zwei 10 GBit Netzwerkschnittstellen ausgerüstet.

Komponente	technische Beschreibung
CPU	2 x Intel Xeon Processor E5-2670 v2, je 10-Core (je 20-Core mit HT), 3,5-3,9 GHz
RAM	256 GB DDR3
HDD	(System) 2 x 2 TB Hitachi/HGST Ultrastar 7K4000 in Software RAID 1
Netzwerk	2 x 10 GBit Intel Ethernet Converged Network Adapter X520-SR2 (Glasfaser)

Tabelle 5.3: Technische Daten der übrigen Server des Profi-Hardware-Clusters

5.3 Software

Sowohl auf dem Commodity- als auch auf dem potenteren Profi-Hardware-Cluster kommt Xubuntu 16.04 zum Einsatz. Auf allen Ceph-Servern müssen folgende Dienste installiert sein:

- openssh-server (passwortlos konfiguriert)
- ntp (für die Zeitsynchronisation der Monitore)

Das gesamte Ceph wird mit dem ceph-internen Kommandozeilentool „ceph-deploy“ von einem auszuwählenden Admin-Node aus installiert. Es benötigt zur Einrichtung der Ceph-Dienste root-Rechte. Man kann entweder alle Kommandos als root ausführen (wie in dieser Arbeit geschehen), oder legt einen Nutzer an, der root-Rechte bekommt. Der Nutzernamen darf nicht „ceph“ lauten, da dieser Name bereits von Ceph reserviert ist. Auf allen Ceph-Servern müssen folgende Befehle ausgeführt werden:

Listing 5.1: Einrichten eines Nutzers für Ceph

```
# Nutzer erstellen
root@kokos:~ > useradd -d /home/testnutzer -m testnutzer

# Passwort festlegen
5 root@kokos:~ > passwd testnutzer

# Nutzer mit sudo-Rechten ausstatten
root@kokos:~ > echo "testnutzer ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/
    testnutzer

10 # Leserechte setzen
root@kokos:~ > chmod 0440 /etc/sudoers.d/testnutzer
```

Das Kommandozeilentool „ceph-deploy“ nutzt ssh, um mit den Servern und Clients eines Ceph-Clusters zu kommunizieren und dort bspw. Ceph zu installieren. Um die benötigten, passwortlosen SSH-Verbindungen zu ermöglichen, muss ein Schlüssel generiert und auf den Ceph-Servern verteilt werden. Auf einem der Rechner, der als Admin-Node dient, muss zur Ausführung kommen:

Listing 5.2: SSH-Key erstellen und verteilen

```

# Schlüsselpaar erstellen
root@kokos:~ > ssh-keygen

4 # öffentlichen Schlüssel verteilen
root@kokos:~ > ssh-copy-id testnutzer@kokos
root@kokos:~ > ssh-copy-id testnutzer@koala
root@kokos:~ > ssh-copy-id testnutzer@cobalt
root@kokos:~ > ssh-copy-id testnutzer@thulium
9 root@kokos:~ > ssh-copy-id testnutzer@thorium

```

Damit man dem Befehl „ceph-deploy“ nicht immer den Parameter „-username {username}“ mitgeben muss, empfiehlt es sich, die Datei `/.ssh/config` zu editieren und dort die betreffenden Rechner einzutragen.

Listing 5.3: SSH-Tuning

```

1 Host kokos
    Hostname kokos
    User testnutzer
Host koala
    Hostname koala
    User testnutzer
6 Host cobalt
    Hostname cobalt
    User testnutzer
Host thulium
11 Hostname thulium
    User testnutzer
Host thorium
    Hostname thorium
    User testnutzer

```

Die Installation von Ceph erfolgt mit dem Ceph-Tool „ceph-deploy“. Damit werden im Folgenden auch die hauptsächlichen Bestandteile Ceph, die OSDs, Monitore und MDS auf den fünf Ceph-Servern eingerichtet. Auf dem schon festgelegten Admin-Server wird „ceph-deploy“ installiert:

Listing 5.4: Installation von ceph-deploy auf dem Admin-Server

```

# Public-Key herunterladen, um die Echtheit der Ceph-Pakete sicherstellen zu können
root@kokos:~ > wget -q -O- 'https://download.ceph.com/keys/release.asc' | sudo apt-key
    add -

# Eintragen der Ceph-Paketquellen in die entsprechende Datei für apt
5 root@kokos:~ > echo deb https://download.ceph.com/debian-jewel/ $(lsb_release -sc)
    main | sudo tee /etc/apt/sources.list.d/ceph.list

# Update der Paketquellen und Installation von ceph-deploy
root@kokos:~ > sudo apt-get update && sudo apt-get install ceph-deploy

```

Nun kann man Ceph auf den Ceph-Servern verteilen. Die zentrale Konfigurationsdatei „ceph.conf“ und der Authentifikations-Keyring benötigen ein eigenes Verzeichnis, dass an beliebiger Stelle angelegt werden kann. Nach der Installation findet man diese Dateien dort. Die Installation von Ceph geschieht weiter, indem auf dem Admin-Node zunächst ein Cluster erstellt wird. Dann wird Ceph auf allen angeschlossenen Ceph-Servern installiert und schließlich der erste Monitor erzeugt, der die Cluster-Map enthält und damit Ansprechpartner für alle an Ceph angebundenen Rechner ist.

Listing 5.5: Installation von Ceph

```
# Anlegen eines zentralen Verzeichnisses für Auth-Keys und die Konfigurationsdatei
    ceph.conf
2 root@kokos:~ > mkdir my-cluster
root@kokos:~ > cd my-cluster/

# Erstellen der ceph.conf auf dem Node, auf dem der erste Monitor installiert werden
    soll.
root@kokos:~ > ceph-deploy new kokos
7
# Installation von Ceph in der Version jewel auf allen Ceph-Servern
root@kokos:~ > ceph-deploy install --release jewel kokos koala cobalt thulium thorium

# Einrichten des ersten Monitors. Anschließend sind im Verzeichnis my-cluster Auth-
    Keys verfügbar.
12 root@kokos:~ > ceph-deploy mon create-initial
```

Ein Ceph-Cluster benötigt Festplatten bzw. OSDs, um seiner Funktion gerecht zu werden. Nach der Installation ergibt die Überprüfung der Cluster-Gesundheit mit dem Kommando „ceph -s“ zunächst, dass dem Cluster noch OSDs fehlen, um arbeiten zu können. Sie müssen im nächsten Schritt hinzugefügt werden. Das Hinzufügen geschieht in drei Schritten¹. Zunächst werden alle verfügbaren Devices auf den Ceph-Servern ermittelt, dann die bestehenden Daten (inkl. MBR und GPT) auf den gewünschten Devices gelöscht und schließlich Dateisysteme darauf geschrieben und die Einbindung in den Ceph-Cluster vorgenommen. Dabei erhält jedes Device ein, seiner Größe gemäßes, Gewicht, das in der Cluster-Map eingetragen wird und für die Balance der Datenobjekte im Cluster nötig ist.

¹Zumindest unter Nutzung von „ceph-deploy“. In den Ceph-Docs sind umfangreichere Wege beschrieben, die etwa bei der Nutzung von Konfigurationsmanagern wie puppet oder ansible nötig werden.

Listing 5.6: Hinzufügen von OSDs

```
# zeigt alle verfügbaren Festplatten auf kokos
root@kokos:~ > cd my-cluster
3 root@kokos:~ > ceph-deploy disk list kokosm koala cobalt

# löscht die vorhandenen Daten der Festplatten /dev/sdb /dev/sdc und /dev/sdd auf
  kokos, koala, cobalt, thulium und thorium (inkl. MBR und GPT)
root@kokos:~ > ceph-deploy disk zap kokos:sdb kokos:sdc kokos:sdd koala:sdb koala:sdc
  koala:sdd cobalt:sdb cobalt:sdc cobalt:sdd thulium:sdb thulium:sdc thulium:sdd
  thorium:sdb thorium:sdc thorium:sdd

8 # partitioniert die Festplatten, erstellt Dateisysteme und bindet die OSDs in Ceph ein
root@kokos:~ > ceph-deploy osd create kokos:sdb kokos:sdc kokos:sdd koala:sdb koala:
  sdc koala:sdd cobalt:sdb cobalt:sdc cobalt:sdd thulium:sdb thulium:sdc thulium:sdd
  thorium:sdb thorium:sdc thorium:sdd
```

Damit ist der Cluster arbeitsfähig.

Da Ceph ausfallsicher sein soll, darf es keinen Single-Point-of-Failure geben. Um dieses Ziel zu erreichen, sollten noch mindestens zwei weitere Monitore erstellt werden. Scheitert deren Erstellung, muss in der „ceph.conf“ das Public-Netzwerk definiert werden. (public_network = xxx.xxx.xxx.xxx/xx). Des weiteren wird mindestens ein MDS benötigt, um das Ceph-Filesystem benutzen zu können. Um wiederum keine einzelne Schwachstelle im Cluster zu haben, werden derer zwei erstellt.

Listing 5.7: Hinzufügen von Monitoren und MDS

```
1 # Erstellen zweier Monitore
root@kokos:~ > ceph-deploy mon create koala
root@kokos:~ > ceph-deploy mon create cobalt

# Erstellen zweier MDS
6 root@kokos:~ > ceph-deploy mds create kokos
root@kokos:~ > ceph-deploy mds create koala
```

Kapitel 6

Benchmarking-Tools

Ein wesentliches Ziel dieser Arbeit ist die prinzipielle Leistungsfähigkeit von Ceph zu bestimmen und es zu optimieren. Daher ist es nötig, Schreib- und Lesegeschwindigkeiten zu messen, um die verschiedenen Hard- und Softwarekonfigurationen bewerten zu können. Rohe Geschwindigkeitsmessungen der verbauten Festplatten werden mit „bonnie++“, der maximale Durchsatz der Netzwerkschnittstellen mit „iperf“ gemessen. Die Schreib- und Lesegeschwindigkeit des Clusters bzw. einzelne seiner Funktionen werden sowohl mit einem Ceph-internen Werkzeug „rados-bench“ als auch mit „fio“ getestet.

6.1 Festplattengeschwindigkeit

Ein begrenzender Faktor der Schreib- und Leseraten eines Ceph-Clusters ist die Geschwindigkeit der für seine OSDs verbauten Festplatten. Daher empfiehlt es sich, deren Schreib- und Leseperformance vor den Testungen zu vermessen. Dies wurde mit dem Unix Kommandozeilenwerkzeug „bonnie++“ [21] ausgeführt.

Mit „bonnie++“ reicht ein einfacher Befehl um Schreib- und Lesegeschwindigkeit mit einem Mal zu testen:

Listing 6.1: Ermitteln der Schreib- und Lesegeschwindigkeit einer Festplatte mit bonnie++

```
# Bsp. für osd.0
root@kokos:~ > bonnie++ -d /var/lib/ceph/osd/ceph-0/ -s 16384 -n 0 -m bonnie_test -f -
b -u root -r 8192
```

6.1.1 Commodity-Hardware-Cluster

Für den Commodity-Hardware-Cluster wurden explizit Platten verschiedener Hersteller und Größe verbaut (Produktbezeichnungen der Festplatten siehe Kapitel 5.1.1, Tabelle 5.1 auf Seite 43), um zu zeigen, dass Ceph anspruchslos ist, was Festplatten angeht und einfach darauf funktioniert. Die erhaltenen Messdaten spiegeln diese Entscheidung wieder. Ceph wäre sogar so flexibel, auch mit Loopback-Devices, also Dateien, die als Laufwerk gemountet werden, klarkommen zu können, doch soll dies hier nicht überprüft werden.

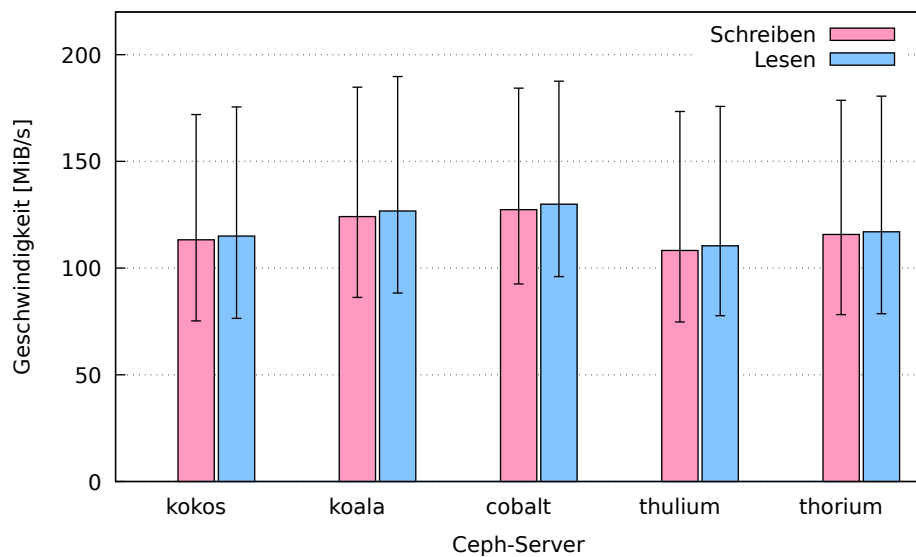


Abbildung 6.1: Durchschnittliche Schreib- u. Leseraten der Festplatten (je 3 pro Ceph-Server)

Die roten und blauen Balken geben das arithmetische Mittel der Geschwindigkeiten aus allen Platten an. Die dünnen Fehlerbalken stellen dar, wie stark die Werte unter den getesteten Festplatten schwankten. Die starken Geschwindigkeitsunterschiede rühren u.a. daher, dass Platten verschiedener Hersteller und Größe verbaut worden sind (siehe Tabelle 5.1 auf Seite 43), deren Leistungsfähigkeit sich sehr voneinander unterscheidet. Da Ceph verspricht, gerade auf heterogener Commodity-Hardware zu laufen, ist dieser Effekt durchaus gewollt.

6.1.2 Profi-Hardware-Cluster

Auf dem Profi-Hardware-Cluster ergibt sich ein anderes Bild, da die hier verbauten Platten (siehe Tabelle 5.2 auf Seite 44) identisch sind. Die Werte zeigen keine Ausreißer, bis auf eine Platte in osceph5, die aus unbekannten Gründen schneller war.

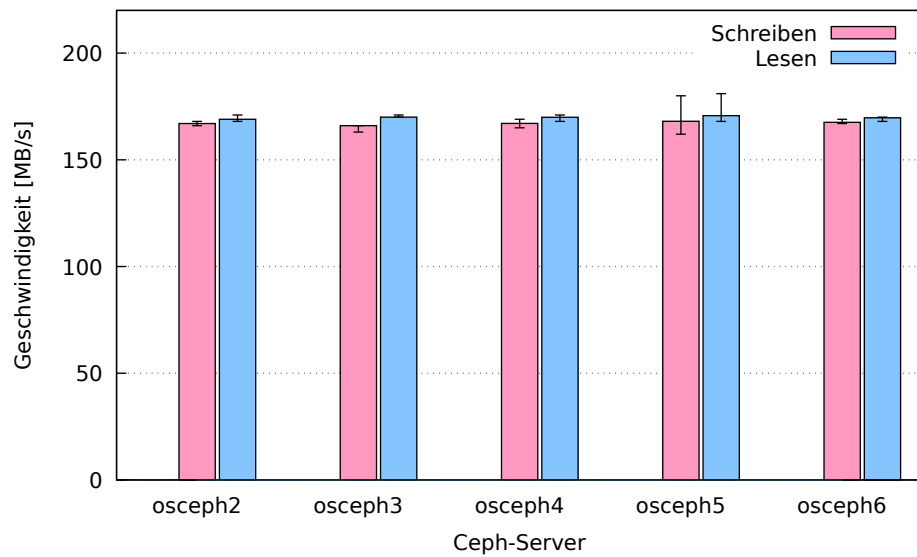


Abbildung 6.2: Durchschnittliche Schreib- u. Leseraten der Festplatten (je 12 pro Ceph-Server)

6.2 iperf

Ein weiterer, die Cluster Performance beeinflussender, Schlüsselwert ist der Netzwerkdurchsatz. Um ihn zu messen, bietet sich das Tool „iperf“ an, das eine Client-Server Verbindung nutzt, um die TCP- und UDP-Bandbreite zwischen zwei Netzwerkteilnehmern zu messen. Da Ceph für seine Netzwerkkommunikation TCP benutzt, wird in dieser Messung die TCP-Geschwindigkeit gemessen. Grundsätzlich wird mit iperf die Geschwindigkeit zwischen zwei Rechnern gemessen, wobei einer als Server, der andere als Client dient. Die Tests werden an dieser Stelle unidirectional ausgeführt und zwar derart, dass der Client sendet und der Server empfängt. Während der Messungen dient ein einziger Rechner als Client, auf den übrigen laufen die Server-Prozesse.

Die Messung geschieht in zwei Schritten. Zuerst wird sie sukzessive ausgeführt, um zu zeigen, wie hoch die maximale Bandbreite einer Verbindung ist. Anschließend werden die Messungen zwischen allen Servern und dem Client gleichzeitig ausgeführt.

Der Grund für die zweite Art der Messung liegt darin, dass der Profi-Hardware-Cluster über getrunzte Interfaces verfügt, die ihre volle Leistungsfähigkeit nur bei mehreren gleichzeitigen Verbindungen ausschöpfen können. Bei ihnen kommt, gemäß IEEE 802.3ad [22], ein Algorithmus zum Einsatz, der per Hash über die Ziel- und Quell-MAC-Adresse eines Paketes darüber entscheidet, über welches Interface der Verkehr geht und so versucht sicherzustellen, dass beide Interfaces gleich stark benutzt werden. In der Summe aller Ein-

zerverbindungen ergibt sich dann der maximale Durchsatz, den der Client mit allen Servern gleichzeitig erreichen kann.

6.2.1 Commodity-Hardware-Cluster

Theoretisch sind alle Rechner des Commodity-Hardware-Clusters untereinander mit einem 1 GBit-Interface angebunden. Die Geschwindigkeit dieser Interfaces wird im Folgenden gemessen. Als Client und Sender dient „kokos“. Von ihm aus werden alle Geschwindigkeiten gemessen.

In folgendem Schaubild wird die Testanordnung abgebildet.

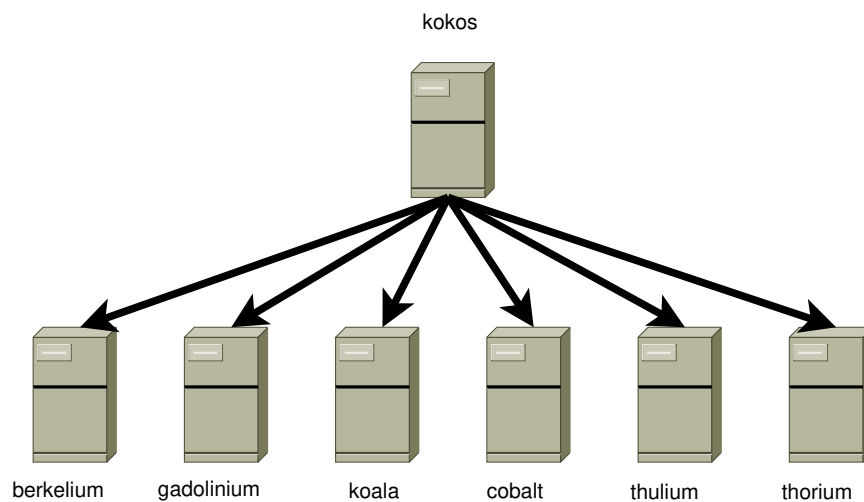


Abbildung 6.3: Testanordnung für iperf-Test

Das Resultat der sukzessiven Messung ergibt, dass bei allen Rechnern nahezu die volle Bandbreite des verbauten 1 GBit-Interfaces erreicht wird.

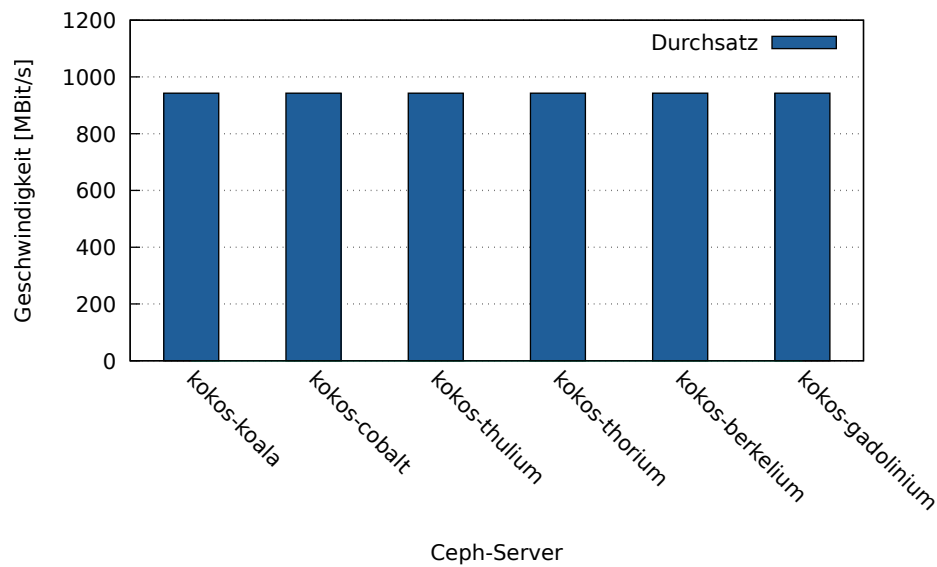


Abbildung 6.4: Netzwerkgeschwindigkeit bei sukzessiver Messung

Wird gleichzeitig gemessen, ergibt die Summe aller Geschwindigkeiten die maximale Geschwindigkeit der Netzwerkschnittstelle des Clients „kokos“, in diesem Falle rund 1 GBit/s.

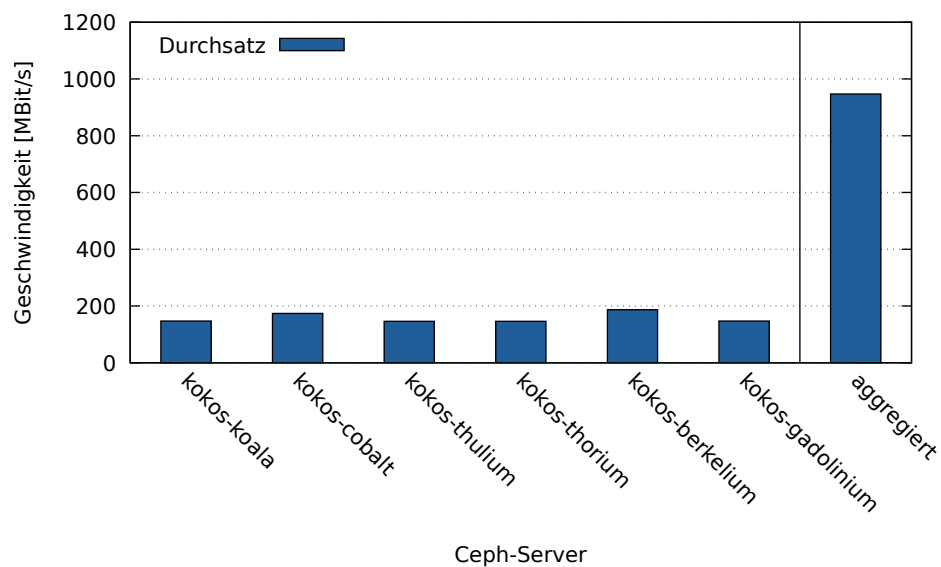


Abbildung 6.5: Netzwerkgeschwindigkeit bei paralleler Messung

6.2.2 Profi-Hardware-Cluster

Theoretisch sind alle Rechner des Profi-Hardware-Clusters untereinander mit zwei getrunken 10 GBit-Interfaces angebunden. Deren Geschwindigkeit wird im Folgenden gemessen.

Als Client und Sender dient „osceph6“. Von ihm aus werden alle Geschwindigkeiten gemessen.

In folgendem Schaubild wird die Testanordnung abgebildet.

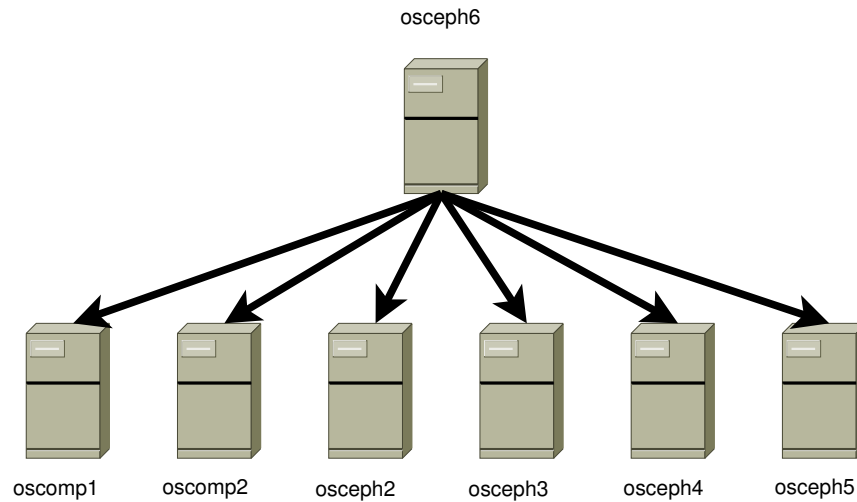


Abbildung 6.6: Testanordnung für iperf-Test

Das Resultat der sukzessiven Messung ergibt, dass bei allen Rechnern nahezu die volle Bandbreite des verbauten LWL 10 GBit-Interfaces erreicht wird.

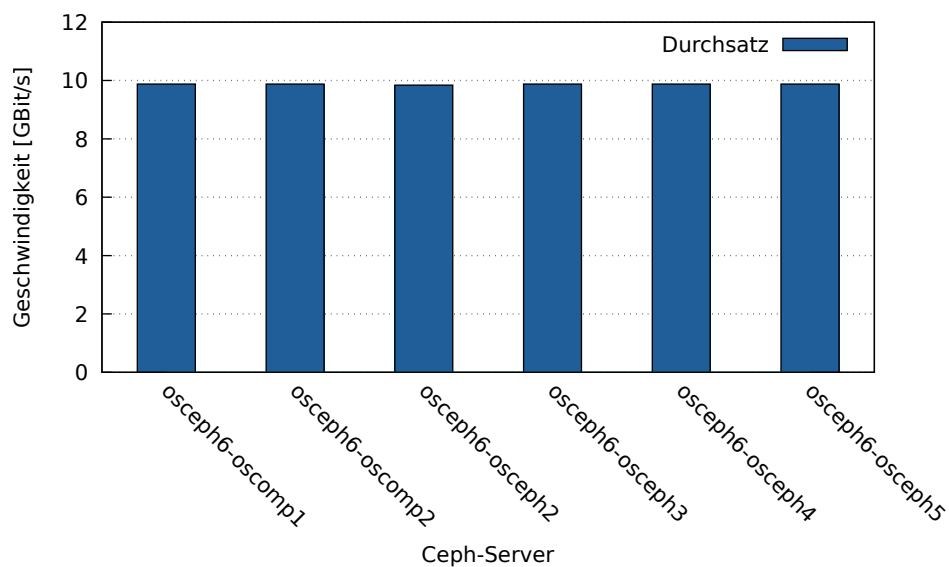


Abbildung 6.7: Netzwerkgeschwindigkeit bei sukzessiver Messung

Wird gleichzeitig gemessen, ergibt die Summe aller Geschwindigkeiten die maximale Geschwindigkeit der gebündelten Netzwerkschnittstelle des Clients „osceph6“, in diesem Falle rund 20 GBit/s. Dies liegt darin begründet, dass die Rechner „oscomp1“, „osceph4“ und „osceph5“ als Ergebnis der genannten Hashfunktion, nur mit einer der beiden Netzwerkin-
terfaces von „osceph6“ kommunizieren und „oscomp2“ allein mit der zweiten.

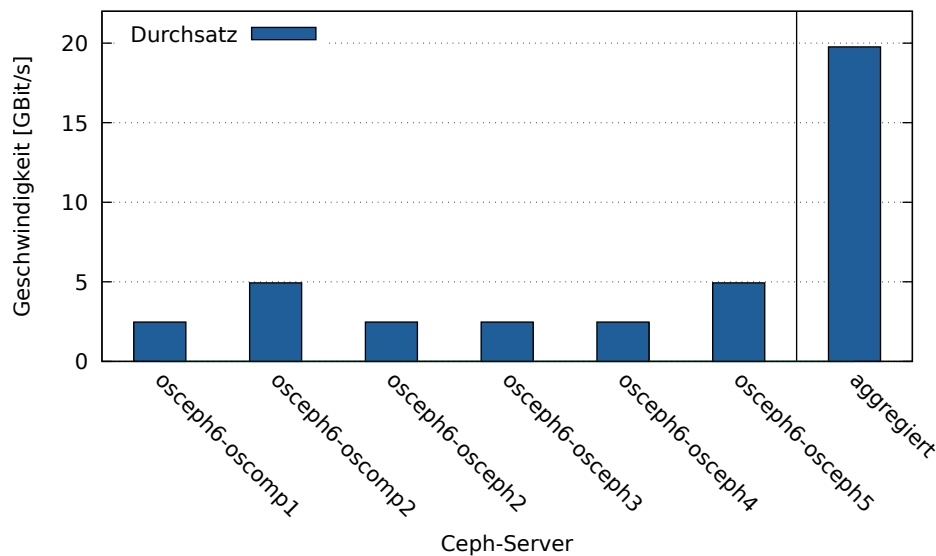


Abbildung 6.8: Netzwerkgeschwindigkeit bei paralleler Messung

6.3 Ceph-interner Benchmark

Ceph bringt einen Benchmark zum Messen der maximalen Schreib- und Lesegeschwindigkeit eines Pools mit: „rados-bench“ [23]. Dieser erlaubt es, sequenzielles Schreiben und Lesen, sowie das wahlfreie Lesen (random) auf dem Cluster zu testen, ohne, dass ein Client angeschlossen sein muss. Der Test wird auf einem Ceph-Server ausgeführt und benötigt nur einen beliebigen Ceph-Pool. Die Syntax des Befehls lautet wie folgt:

Listing 6.2: Ceph-interner Benchmark

```
# starte einen Schreib-Benchmark im Pool test mit 120 Sekunden, 4 MB Blocksize, 16
  Threads; lösche die Daten anschließend nicht
root@kokos:~ > rados bench -p test 120 write -b 4M -t 16 --no-cleanup
3
# starte einen Lese-Benchmark im Pool test mit 120 Sekunden, 4 MB Blocksize, 16
  Threads; lösche die Daten anschließend nicht
root@kokos:~ > rados bench -p test 120 seq -b 4M -t 16 --no-cleanup

# starte einen zufälligen Lese-Benchmark im Pool test mit 120 Sekunden, 4 MB Blocksize
  , 16 Threads; lösche die Daten anschließend nicht
8 root@kokos:~ > rados bench -p test 120 rand -b 4M -t 16 --no-cleanup

# lösche die erstellten Dateien wieder
root@kokos:~ > rados -p test cleanup
```

Die Ergebnisse befinden sich in den folgenden Kapiteln für je unterschiedliche und noch zu erklärende Konfigurationen.

6.4 fio

Im Gegensatz zum cephinternen Benchmark „rados-bench“, wird der „Flexible I/O Tester“ „fio“ genutzt, um vom Client aus die Leistung des Systems unter reproduzierbaren und vergleichbaren Bedingungen zu testen. Dabei wird das Tool dafür genutzt, die Schreib- und Lesegeschwindigkeiten eines oder mehrerer Clients unter wechselnden Konfigurationen des Clusters zu testen, um herauszufinden, unter welchen Bedingungen er optimal arbeitet.

Fio unterstützt mehrere I/O-Engines wie „libaio“ oder „sync“ und kann dadurch sehr flexibel eingesetzt werden. Die Wahl der Engine beeinflusst die Art und Weise, in der die Ein- bzw. Ausgabe der Daten durchgeführt werden [24]. Für den vorliegenden Fall konnte die Engine „rbd“ benutzt werden, die speziell dafür gemacht ist, RADOS Block-Devices zu testen [25]. Dabei funktioniert die Engine nur für den Fall des Schreibens. Für Lesezugriffe eignet sie sich nicht. Der Vorteil dieser Engine liegt darin, dass sie den Linux-Page-Cache nicht nutzt und dadurch die pure Schreibgeschwindigkeit, unabhängig von der Größe des clientseitigen Arbeitsspeichers, misst.

Die fio-Benchmarks werden mithilfe von Kommandozeilenparametern oder Konfigurationsdateien definiert und angepasst. So lassen sich individuelle Tests entwerfen, die unterschiedliche Szenarien einfach und detailliert abbilden können [24]. Einstellbare Parameter sind bspw. die Größe der Blocksize, die Gesamtgröße der zu schreibenden oder lesenden Datei, Name und Format der Ausgabedatei, sowie, bei der Engine „rbd“, die Bezeichnung des Block-Devices, das es zu testen gilt. Die Parameter, die für die in dieser Arbeit durchgeführten Tests mit fio benutzt wurden, lauten:

--name	Name des Tests und der anzulegenden Dateien
--ioengine	Zu verwendende I/O-Engine
--rw	Art des Tests (read/write/randread/randwrite)
--bs	Blockgröße
--numjobs	Anzahl paralleler Jobs
--size	Dateigröße pro Job
--output	Pfad zur Ausgabedatei
--minimal	Ausgabe im terse-Format
--kb-base	Kilobytes oder Kibibytes als Basis
--rbdname	Name des RADOS Block-Devices
--clientname	Name des Ceph-Nutzers
--pool	Name des Ceph-Pools
--runtime	zeitliche Länge des Tests

Tabelle 6.1: Kommandozeilenparameter von fio [4]

Fio-Ausgabedateien sind maschinenlesbar und können mit Hilfe von Bash-Kommandos ausgewertet werden. Mithilfe des fio-Parameters „--minimal“ geschieht die Ausgabe der Dateien zudem in Form einer einzeiligen CSV-Datei („terse“-Output genannt), die sehr einfach mittels „cut“ weiterverarbeitet werden kann. Im anschließenden Listing befindet sich die Ausgabe einer solchen Datei. Die in KB/s angegebene durchschnittliche Lesegeschwindigkeit befindet sich bspw. an Position 7. Im Internet finden sich Übersichten aus denen man entnehmen kann, wie die CSVs aufzulösen sind [26]. Auf den Seiten der Fio-Dokumentation konnten diese Informationen unerfreulicher Weise nicht gefunden werden.

Listing 6.3: fio Beispiel für CSV-Ausgabe

```

3; fio-2.2.10; bench_fio_rbd-Iter-1-berkelium-rbd0-randread-NumJobsForRead-4-BS-16M;
0; 0; 8388608; 82005; 5; 102293; 0; 0; 0.000000; 0.000000; 228357; 2252336; 777290.394531;
321019.144963; 1.000000%=280576; 5.000000%=342016; 10.000000%=411648; 20.000000%=497664;
4 30.000000%=585728; 40.000000%=643072; 50.000000%=716800; 60.000000%=806912;
70.000000%=905216; 80.000000%=1036288; 90.000000%=1220608; 95.000000%=1335296;
99.000000%=1662976; 99.500000%=1892352; 99.900000%=2244608; 99.950000%=2244608;
99.990000%=2244608; 0%=0; 0%=0; 0%=0; 0%=0; 228358; 2252337; 777291.322266; 321019.240696;
7275; 53281; 27.459212%; 22517.926887; 7680.509045; 0; 0; 0; 0; 0; 0; 0.000000; 0.000000; 0; 0;
9 0.000000; 0.000000; 1.000000%=0; 5.000000%=0; 10.000000%=0; 20.000000%=0; 30.000000%=0;
40.000000%=0; 50.000000%=0; 60.000000%=0; 70.000000%=0; 80.000000%=0; 90.000000%=0;
95.000000%=0; 99.000000%=0; 99.500000%=0; 99.900000%=0; 99.950000%=0; 99.990000%=0;
0%=0; 0%=0; 0%=0; 0; 0; 0.000000; 0.000000; 0; 0; 0.000000; 0.000000; 0.000000; 0.003015%;
1.730740%; 6676; 0; 283; 100.0%; 0.0%; 0.0%; 0.0%; 0.0%; 0.0%; 0.0%; 0.0%; 0.0%; 0.0%;
14 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.00%; 0.39%;
19.73%; 32.62%; 24.80%; 22.07%; 0.39%; rbd0; 6654; 10; 6134; 17; 391348; 440; 391808; 99.96%

```

Da es sich bei Ceph um ein verteiltes System handelt, müssen Zugriffe von mehreren Clients aus simuliert werden, um herauszufinden, wie sich die Schreib-/Leseraten bei parallelem Zugriff verhalten. Die Testanordnung gestaltete sich so, dass die fio-Prozesse auf mehreren Clients gleichzeitig gestartet wurden. Dabei wurde die Anzahl paralleler fio-Prozesse ge-

benenfalls erhöht oder verringert, um gleichzeitige Zugriffe zu simulieren. Die entscheidende Größe bei diesen Zugriffen spielt das Netzwerk. Wenn dessen Kapazität ausgeschöpft ist, verändert auch die Vergrößerung der Anzahl gleichzeitig zugreifender Prozesse die Leistung nicht mehr.

Da im Verlaufe dieser Arbeit sowohl die RADOS Block-Devices, als auch das Ceph-Filesystem getestet werden sollen, wurden zwei verschiedene fio-Engines verwendet. Zum Einen die Standardengine „sync“ für das Ceph-Filesystem und die Lesezugriffe auf das RADOS Block-Device und zum Anderen die Engine „rbd“ für das Messen der Schreibgeschwindigkeit auf RADOS Block-Devices. Im Folgenden seien die in dieser Arbeit verwendeten fio-Kommandos abgebildet.

Für das Messen mit der Engine „sync“:

Listing 6.4: fio-Kommando für Ceph-Filesystem und Lesen von RADOS Block-Devices

```
# zum Schreiben (für zufällige Schreibzugriffe ggfs. mit --rw=randwrite)
fio --name=cephfs --rw=write --size=1T --bs=4M --directory=/mnt --output=cephfs_test.
    log --minimal --kb_base=1024 --ioengine=sync --numjobs=1

# zum Lesen (für zufällige Lesezugriffe ggfs. mit --rw=randread)
5 fio --name=cephfs --rw=read --size=1T --bs=4M --directory=/mnt --output=cephfs_test.
    log --minimal --kb_base=1024 --ioengine=sync --numjobs=1
```

Für das Messen mit der Engine „rbd“:

Listing 6.5: fio-Kommando für das Schreiben auf RADOS Block-Devices

```
# zum Schreiben
fio --name=cephrbd --ioengine=rbd --clientname=admin --pool=test --rw=write --runtime
    =120 --bs=4M --output=cephrbd_test.log --minimal --kb_base=1024 --rbdname=
    test_device
```

Die Ergebnisse befinden sich in den folgenden Kapiteln für je unterschiedliche und noch zu erklärende Konfigurationen.

Kapitel 7

Benchmarks nach Standardinstallation

Nachdem die beiden Ceph-Cluster mithilfe des Kommandozeilentools „ceph-deploy“ in der Version 10.2.0 „Jewel“ (major release 04/2016) installiert wurden, befinden sie sich im einfachsten Standardzustand, ohne etwaige Optimierungen. D.h. es liegt der Filestore als Speicherbackend vor, die OSDs sind auf den eingebauten Festplatten installiert und die Journale (siehe Abbildung 2.3 auf Seite 17) befinden sich auch darauf, was Leistung kostet, da diese Platten langsam sind und der gleichzeitige Zugriff auf Journal und OSD Zeit kostet. Des Weiteren werden in dieser Konfiguration replizierte Pools mit der Replikationsgröße von drei verwendet, d.h. von jedem Datenobjekt existieren noch zwei weitere Kopien. In späteren Messungen werden diese Konfigurationsparameter variiert werden und separate SSDs als Journaldatenträger verwendet, um den Cluster zu beschleunigen.

Es folgen Ergebnisse von Messungen des Commodity- und des Profi-Hardware-Clusters nach dieser Standardinstallation.

7.1 Commodity-Hardware-Cluster

Wichtige Konfigurationsdaten:

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Journale auf selber Festplatte wie OSD
- Poolart Replikation
- Replikationszahl drei
- insg. 15 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.1 auf Seite 43)

- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 980 MBit/s
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 585 MB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 575 MB/s

7.1.1 Ceph-interner Benchmark

Der cephinterne Benchmark „rados-bench“ kennt im Wesentlichen nur die Optionen Blocksize und Threads, sowie Schreiben, Lesen (sequenziell) und Lesen (wahlfrei). Es wurden Threadanzahlen von 1 bis 64 getestet, wobei sich jede Threadanzahl aus dem Quadrat der vorherigen ergibt. Die besten Leistungen wurden von 64 Threads erzielt. Bei dieser Anzahl wird die volle Leistungsfähigkeit des Clusters ausgeschöpft. Die folgende Grafik veranschaulicht die Schreib-/Leseleistung des Commodity-Hardware-Clusters pro Sekunde bei verschiedenen Blocksizes und 64 gleichzeitigen Threads.

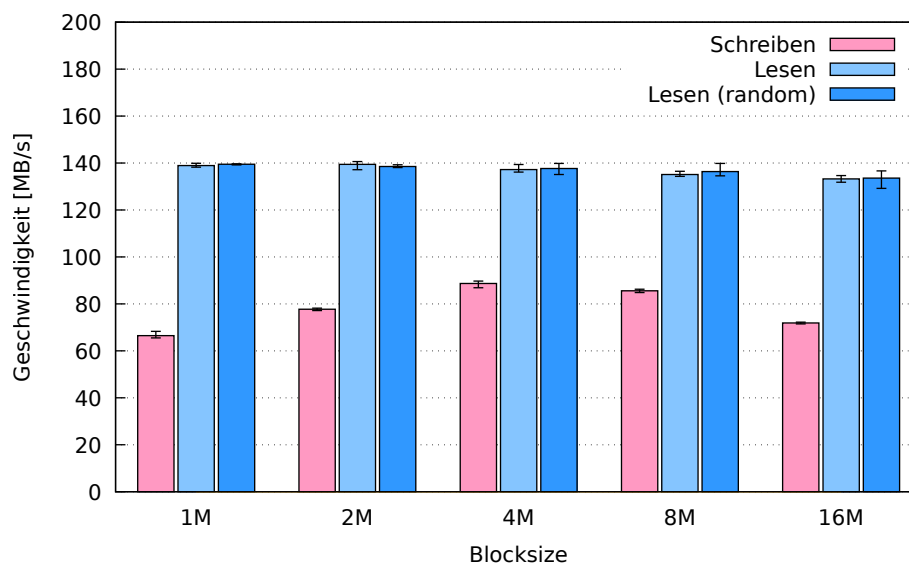


Abbildung 7.1: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

7.1.2 RADOS Block-Device mit fio

Als nächstes wurde von den Clients aus getestet und zwar mit je einem gemounteten Block-Device. Die Tests für massive parallele Zugriffe folgen später in dieser Arbeit. Die Messung der Schreibleistung wurde mit der fio-Engine „rbd“ gemessen, die keine Caching-Effekte erzeugt. Dies ist dadurch belegbar, dass das Senden von Paketen vom Client zum Cluster genau zu dem Zeitpunkt aufhört, zu dem auch der fio-Prozess endet. Während des Designs der Tests zeigte sich, dass sich Cachingeffekte dadurch negativ äußern, dass der fio-Prozess endet, das Senden von Paketen aber noch einige Zeit weiter geht, da die Daten aus dem

Cache des Clients gelesen werden. In diesem nicht gewollten Fall, meldet fio Vollzug, obwohl noch gar nicht endgültig geschrieben wurde.

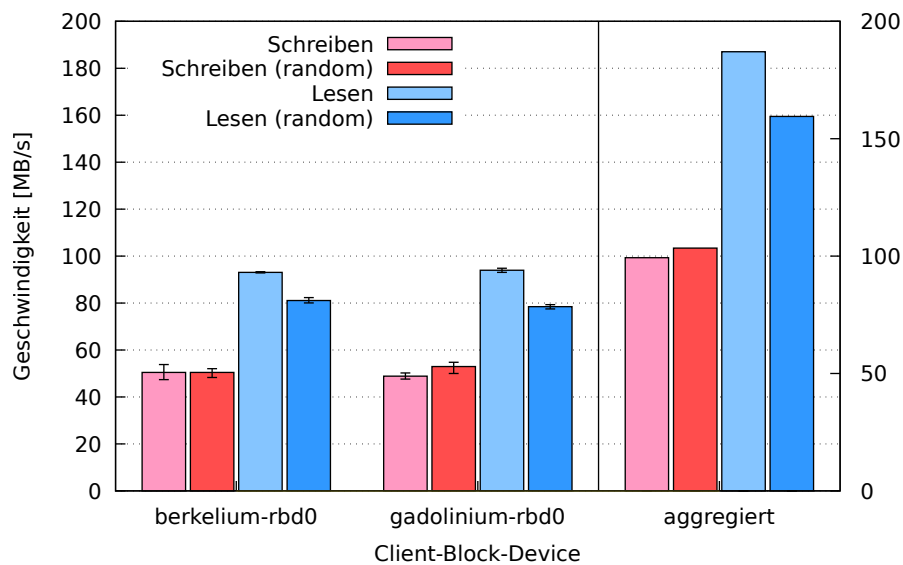


Abbildung 7.2: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

7.1.3 Ceph-Filesystem mit fio

Der Test des Ceph-Filesystems erfolgte ebenfalls von zwei Clients aus. Die Werte unterscheiden sich nur leicht von denen des vorigen Tests. Es sind keine großartigen Performanceunterschiede auszumachen.

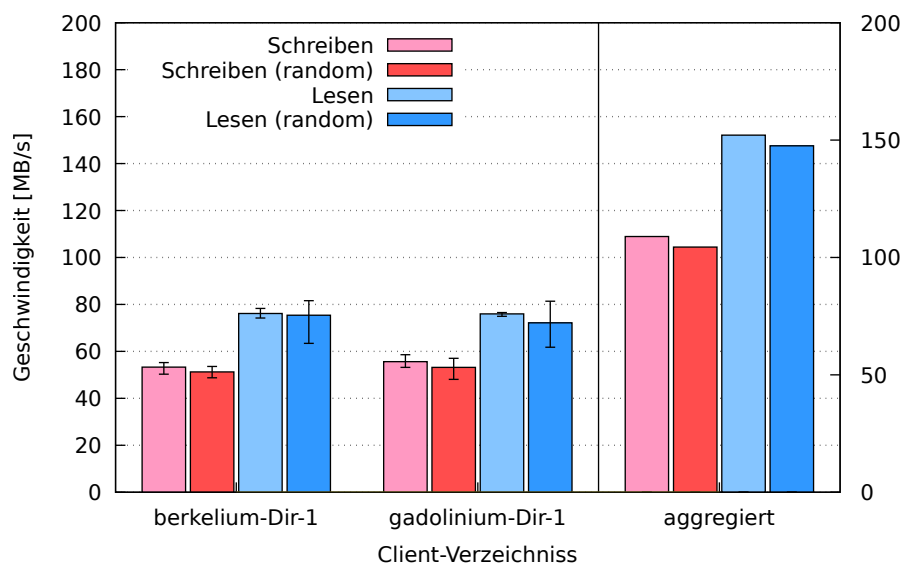


Abbildung 7.3: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

7.1.4 Zusammenfassung

Die Benchmarks dieses initialen Tests ergaben, dass mit einer Geschwindigkeit zwischen ca. 70-90 MB/s im ceph-internen Benchmark und ca. 100 MB/s auf dem RADOS Block-Device und dem Ceph-Filesystem geschrieben werden konnte. Dass der ceph-interne Benchmark teils hinter den beiden fio-Tests zurückbleibt, kann durch eine Beobachtung erklärt werden, die beim Monitoring der Ceph-Server gemacht wurde. Beim ceph-internen Benchmark schreiben jeweils nur vier der Ceph-Server, während einer zum Client wird und die Daten zum Schreiben liefert. Dadurch lässt sich erklären, warum der Cluster bei den externen fio-Tests schneller arbeitete, da in diesem Falle alle fünf Ceph-Server schrieben.

Die Lesewerte liegen zwischen 140 MB/s beim ceph-internen Benchmark und bis zu 185 MB/s beim RADOS Block-Device bzw. 150 MB/s beim Ceph-Filesystem. Die Beschränkung auf diese Geschwindigkeit liegt mit hoher Wahrscheinlichkeit an der limitierten Kapazität der 1 GBit Netzwerkinterfaces, mit dem Clients und Ceph-Server angebunden sind.

7.2 Profi-Hardware-Cluster

Der Profi-Hardware-Cluster befindet sich ebenfalls in der Standardkonfiguration, die nach der Installation vorliegt. Seine Konfigurationsdaten entsprechen denen des kleineren Clusters.

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Journale auf selber Festplatte wie OSD
- Poolart Replikation
- Replikationszahl drei
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

7.2.1 Ceph-interner Benchmark

Die Parameter des durchgeführten Tests unterscheiden sich nicht von denen des Commodity-Hardware-Clusters. Die Leistungsunterschiede sind jedoch enorm. Zu bedenken ist, dass die durchschnittlichen Schreibraten der Festplatten des Profi-Hardware-Clusters (siehe Abbildung 6.1 auf Seite 50 und Abbildung 6.2 auf Seite 51) über denen des Commodity-Hardware-Clusters liegen und die Netzwerkleistung (siehe Abbildung 6.4 auf Seite 53 und Abbildung 6.7 auf Seite 54) wesentlich besser ist. Auch die höhere Prozessorleistung und der größere und schnellere RAM spielen sicherlich eine Rolle.

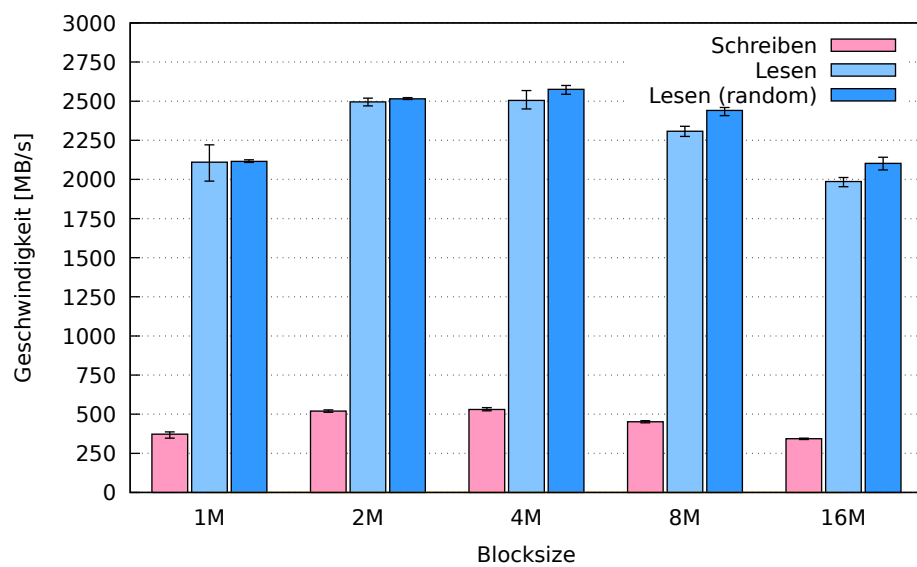


Abbildung 7.4: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

Auch beim Test des RADOS Block-Devices wurden die selben Parameter wie beim Commodity-Hardware-Cluster verwendet. Es gibt beträchtliche Unterschiede zum „kleinen“ Cluster, die auf die angeführten Unterschiede in der Hardwareausstattung zurückzuführen sind.

7.2.2 RADOS Block-Device mit fio

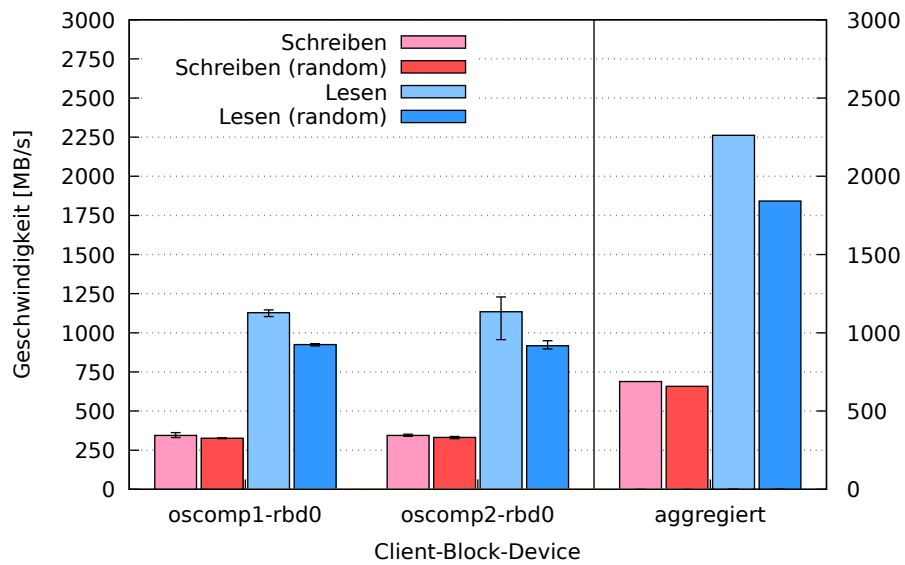


Abbildung 7.5: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Beim Test des Ceph-Filesystems fallen die Unterschiede zum Commodity-Hardware-Cluster ebenso beträchtlich aus. Im Vergleich zum RADOS Block-Device Test auf dem selben Cluster, schneidet das Ceph-Filesystem jedoch schlechter ab.

7.2.3 Ceph-Filesystem mit fio

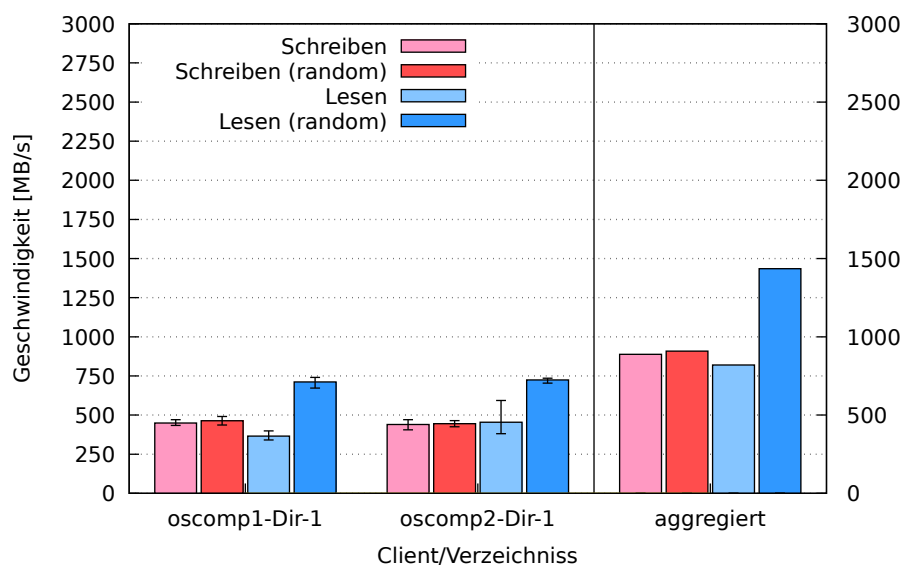


Abbildung 7.6: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

7.2.4 Zusammenfassung

Die Schreibraten beim ceph-internen Benchmark erreichen maximal ca. 550 MB/s und fallen damit wesentlich höher aus, als beim Commodity-Hardware-Cluster. Beim RADOS Block-Device und dem Ceph-Dateisystem sind die Werte vergleichbar gut. Dadurch, dass der Profi-Hardware-Cluster hardware-mäßig in alle Belangen besser ausgestattet ist als sein kleineres Pendant, sind diese Zahlen nicht verwunderlich. Dazu kommt, dass ein grundsätzlich größerer Cluster, mit deutlich mehr OSDs bzw. Festplatten, die Schreibrequests auch effektiver und breiter verteilen kann. Das heißt, es schreiben und lesen mehr Festplatten gleichzeitig.

Die Lesewerte beim ceph-internen Benchmark und dem RADOS Block-Device sind sehr gut und pendeln zwischen 1.800 MB/s und 2.500 MB/s. Allein das Ceph-Filesystem weist schlechtere Werte auf. Diese schlechten Resultate liegen wahrscheinlich an den Operationen, die mit dem Metadaten-Server in Verbindung stehen, da diese Aktionen bei den beiden anderen Tests nicht nötig sind. Außerdem liest das Ceph-Filesystem von zwei Pools gleichzeitig: dem Datenpool und dem Metadatenpool. Es müssen also potentiell mehr Operationen auf dem Cluster ausgeführt werden als bspw. beim RADOS Block-Device. Des Weiteren sollte nicht außer Acht gelassen werden, dass das Ceph-Filesystem das noch jüngste Interface im Ceph-Universum ist und damit noch Raum für Verbesserungen bietet.

In den folgenden Kapiteln wird versucht, die Leistung Ceph durch Hardwareanpassungen und die Variation verschiedener Konfigurationen zu steigern bzw. anzupassen. Dabei wird davon abgesehen, die Maßnahmen auf dem Commodity-Hardware-Cluster durchzuführen, da das Ziel dieser Untersuchung, zu zeigen, dass Ceph auf solcher Art Hardware lauffähig ist, erreicht worden ist. Die Hard- und Softwareanpassungen sowie die weiteren Messungen werden nur noch auf dem stärkeren Profi-Hardware-Cluster durchgeführt.

Kapitel 8

Ansatzpunkte für allgemeine Optimierungen

Nachdem die beiden Ceph-Cluster in der Standardkonfiguration auf ihre Leistungsfähigkeit hin untersucht worden sind, wird nun nur noch der leistungsfähigere Profi-Hardware-Cluster betrachtet.

In den folgenden Abschnitten sollen Umkonfigurierungen vorgenommen werden, um herauszufinden, wie man die Leistung des Ceph-Clusters verbessern kann.

Im Abschnitt 8.1 wird der Filestore als Speicherbackend belassen, Replikationspools genutzt und als Änderung nur SSDs als Journale hinzugefügt. Durch Variation der Replikationszahl wird untersucht, welchen Einfluss diese Größe auf die Geschwindigkeit des Clusters hat.

Im Abschnitt 8.2 wird auf den Bluestore als Speicherbackend gewechselt und zunächst, wie gehabt, Replikationspools untersucht, bei denen die Replikationszahl variiert wird.

Im Abschnitt 8.3 wird wiederum der Bluestore als Speicherbackend verwendet, aber diesmal das Erasure-Coding für die Datenpools verwendet. Bei diesen Messungen wird wieder die Replikationszahl variiert, die hier strengenommen keine Replikationszahl mehr ist, sondern die Anzahl der Coding-Chunks, die aber ebenso Auskunft darüber gibt, wieviele Ceph-Server ausfallen können, bevor ein Datenobjekt verloren geht.

8.1 Filestore, SSDs, Replikation

Als entscheidender Punkt in der Umkonfigurierung der Hardware ist die Auslagerung der Journale auf separate SSDs anzusehen, wodurch sich vor allem die Schreibleistung des Clusters verbessern soll. In der Standardkonfiguration eines Ceph-Clusters ohne separaten Journalen auf SSDs, befinden sich die Journale auf den selben physischen Laufwerken wie

die OSDs, auf denen die Datenobjekte gespeichert werden. Das kostet selbstverständlich Leistung, weil diese Laufwerke in der Regel langsame Festplatten sind und sich OSD und Journal im Betrieb gegenseitig stören. Der Schreib-/Lesekopf muss im ungünstigsten Fall beständig zwischen OSD und Journal hin und her wandern.

Um die Umkonfigurierung der Hardware vorzunehmen, wurde der Cluster komplett neu aufgesetzt und die OSDs und Journale mit folgendem Befehl [27] eingerichtet (Beispiel für zwei OSDs und zwei Journale auf einer SSD; Für weitere OSDs und SSDs müssen nur entsprechend mehr Devices angegeben werden.):

Listing 8.1: Einrichten von OSDs mit separaten Journalen

```
# Formatieren der Laufwerke /dev/sde und /dev/sdf
root@osceph6:~ > ceph-deploy disk zap osceph6:sde osceph6:sdf
3
# Vorbereiten der OSDs und Assoziation mit der SSD /dev/sdc
root@osceph6:~ > ceph-deploy osd prepare osceph6:sde:/dev/sdc osceph6:sdf:/dev/sdc

# Integration der OSDs und zweier Partitionen der SSD in den Ceph-Cluster
8 root@osceph6:~ > ceph-deploy osd activate osceph6:/dev/sde:/dev/sdc1 osceph6:/dev/sdf
:/dev/sdc2
```

Nachdem diese Aktion für alle Ceph-Server, OSDs und SSDs vorgenommen wurde, steht der umkonfigurierte Cluster bereit und kann benutzt werden. Im Folgenden werden nun die gleichen Tests wie vor dem Umbau ausgeführt, um herauszufinden, ob sich eine Verbesserung der Schreib-/Lesegeschwindigkeiten ergeben hat.

In den folgenden Tests soll der Cluster mit den Replikationszahlen eins bis drei getestet werden, um herauszufinden, welchen Einfluss diese Größe auf dessen Leistung hat. Die Replikationszahl legt fest, wie viele Exemplare eines Objekts erstellt und auf dem Cluster verteilt werden sollen. Liegt sie bspw. bei eins, existiert nur ein Exemplar eines Datenobjektes im Cluster, liegt sie dagegen bei drei, gibt es derer drei, was Vorteile bei der Ausfallsicherheit mit sich bringt, da in diesem Fall zwei OSDs ausfallen können, ohne die Verfügbarkeit des Objektes einzuschränken.

Folgende Schritte müssen auf einem Ceph-Server unternommen werden, um einen Pool zu erstellen, der bspw. die Replikationszahl eins hat:

Listing 8.2: Erstellen eines Pools mit der Replikationszahl eins

```
# Erstellen eines Pools test mit PG-Anzahl 8196
2 root@osceph6:~ > ceph osd pool create test 8196 8196

# Setze die Replikationszahl des Pools test auf eins
root@osceph6:~ > ceph osd pool set test size 1
```

8.1.1 Replikationszahl 1

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Pro Ceph-Server zwei SSDs mit je 6 Journalen
- Poolart Replikation
- Replikationszahl eins
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, ausgeführt um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

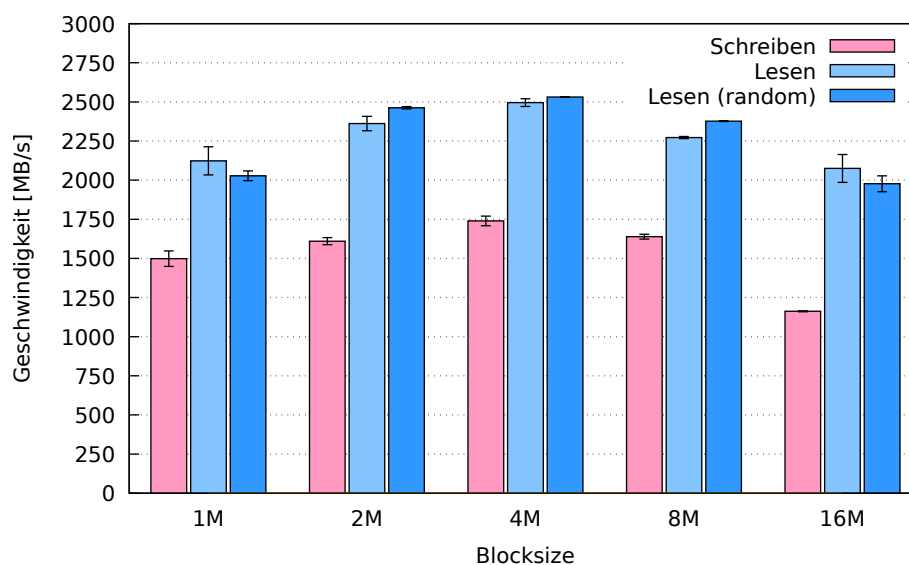


Abbildung 8.1: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

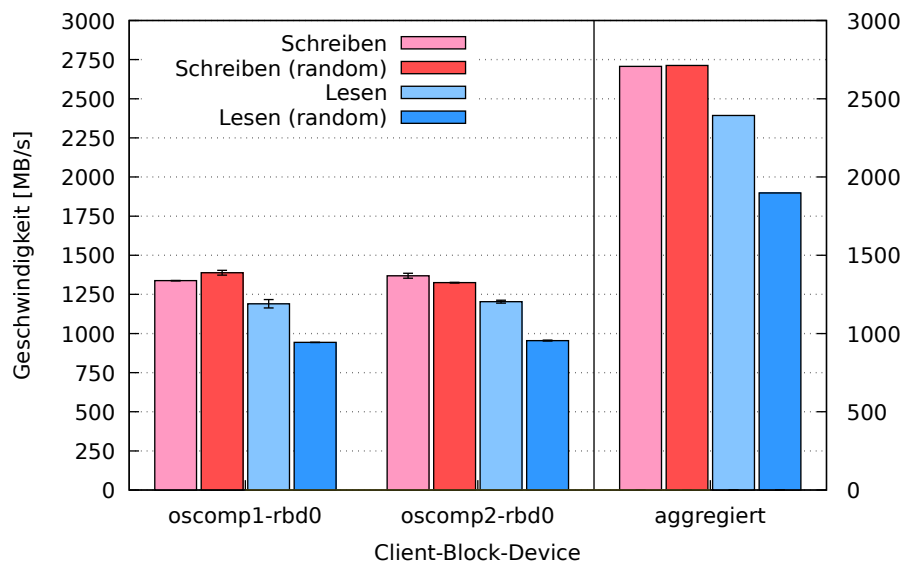


Abbildung 8.2: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

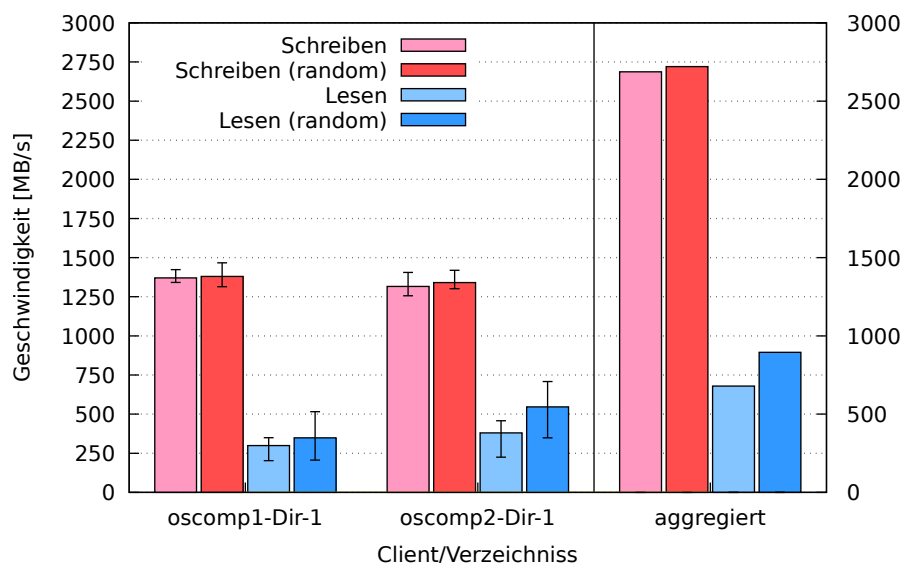


Abbildung 8.3: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

8.1.1.1 Zwischenergebnis

Alle drei Benchmarks zeigen recht hohe Schreibwerte zwischen ca. 1180 MB/s (langsamster Wert beim cephinternen Benchmark) und ca. 2750 MB/s (summierter Wert beim Block-Device und Ceph-Filesystem), was darauf zurückgeführt werden kann, dass jeweils nur eine Kopie (zudem nur auf das Journal einer SSD) gespeichert werden muss, bevor der Schreibvorgang ein Acknowledgement zurückgibt.

Bei den Lesewerten unterscheiden sich vor allem das Block-Device und das Ceph-Filesystem erheblich, wenn nur eine Replik gelesen werden kann. Die relativ langsamen Werte des Ceph-Dateisystems könnten daran liegen, dass die Abfrage der Metadaten vom Metadatenserver sowie das Lesen von zwei Pools gleichzeitig (siehe Kapitel 7.2.4) eine Rolle spielt, die beim Block-Device ausbleibt. Des Weiteren ist das Dateisystem die jüngste Entwicklung in der Familie der Ceph-Speicherinterfaces und damit noch am wenigsten optimiert. Darauf wird später noch eingegangen werden.

Die hohen Lesewerte des cephinternen Benchmarks können mit der erheblichen Thread-Anzahl begründet werden, mit der gemessen wurde. Sie stellen deshalb das Maximum an möglicher Lesegeschwindigkeit dar. Theoretisch müsste man mit 64 Clients gleichzeitig zugreifen, um sie zu erreichen.

8.1.2 Replikationszahl 2

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Pro Ceph-Server zwei SSDs mit je 6 Journalen
- Poolart Replikation
- Replikationszahl zwei
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, ausgeführt um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

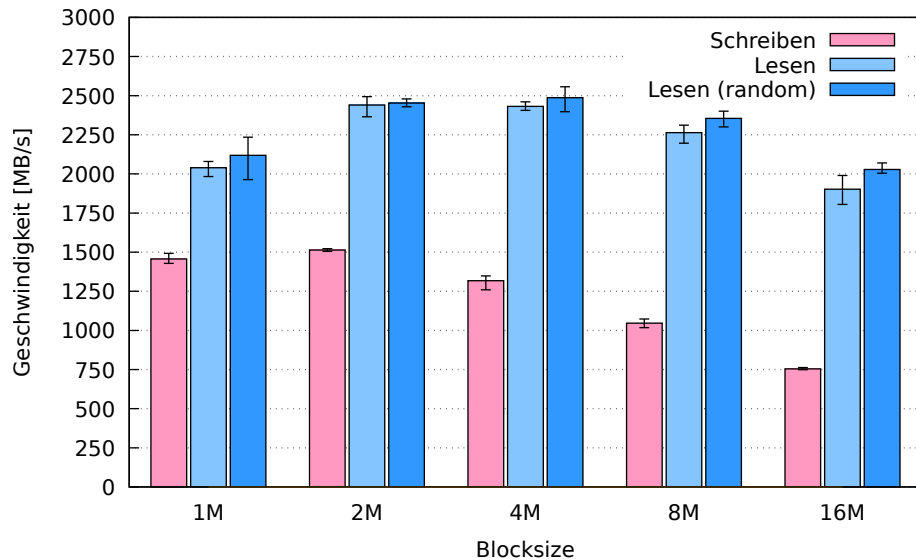


Abbildung 8.4: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

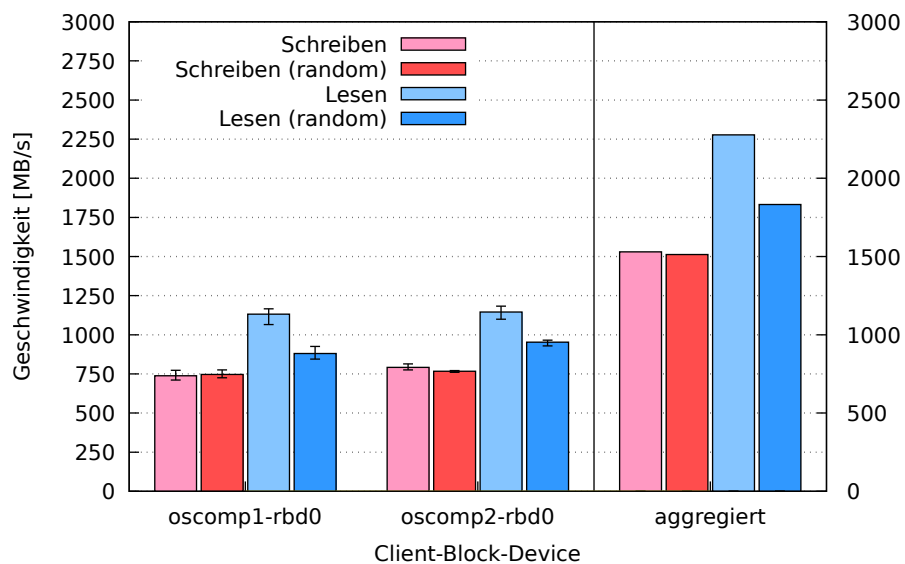


Abbildung 8.5: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

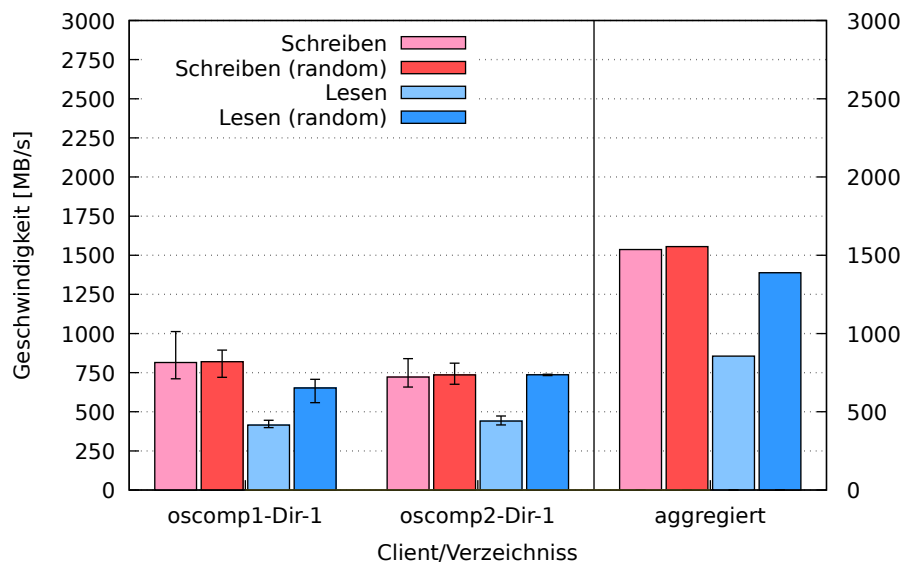


Abbildung 8.6: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

8.1.2.1 Zwischenergebnis

Alle drei Benchmarks zeigen immer noch hohe Schreibwerte zwischen ca. 750 MB/s (langsamster Wert beim cephinternen Benchmark) und ca. 1500 MB/s (summierter Wert beim Block-Device und Ceph-Filesystem). Da nur zwei Kopien geschrieben werden müssen, bevor ein Acknowledgement zurückgegeben wird, können diese Werte so hoch ausfallen.

Bei den Lesewerten unterscheiden sich Block-Device und Ceph-Filesystem wieder. Das Letztere scheint bei Leseoperationen grundsätzlich schlechter zu funktionieren, möglicherweise aus den oben genannten Gründen.

8.1.3 Replikationszahl 3

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Pro Ceph-Server zwei SSDs mit je 6 Journalen
- Poolart Replikation
- Replikationszahl drei
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)

- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, ausgeführt um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

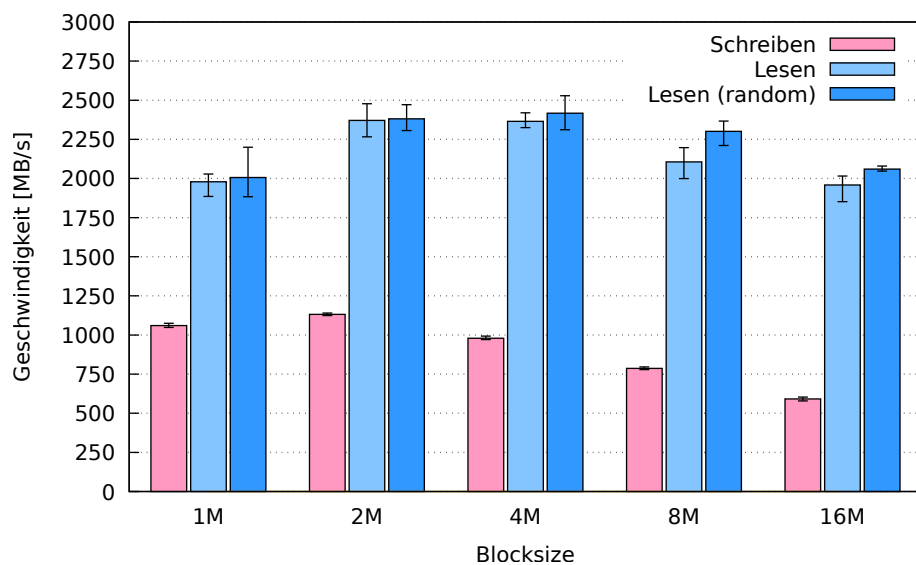


Abbildung 8.7: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

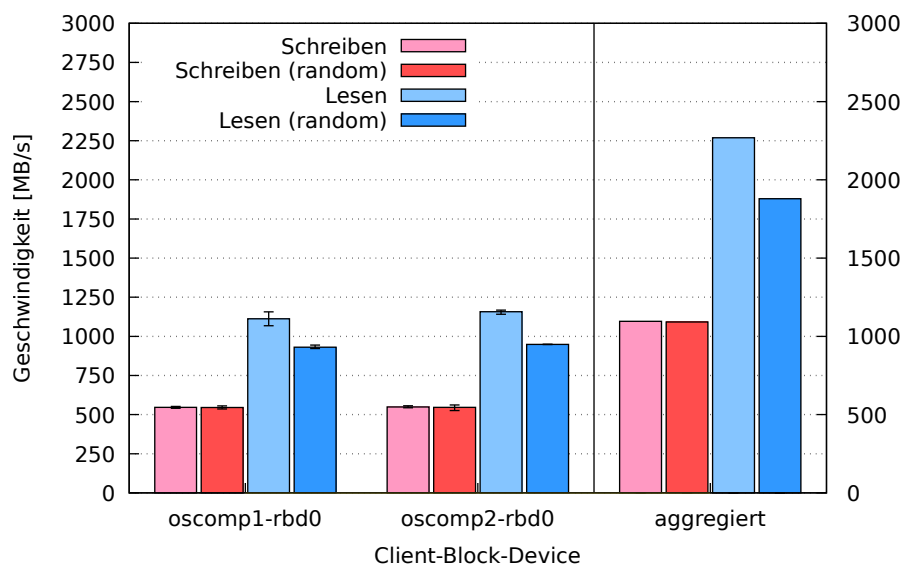


Abbildung 8.8: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

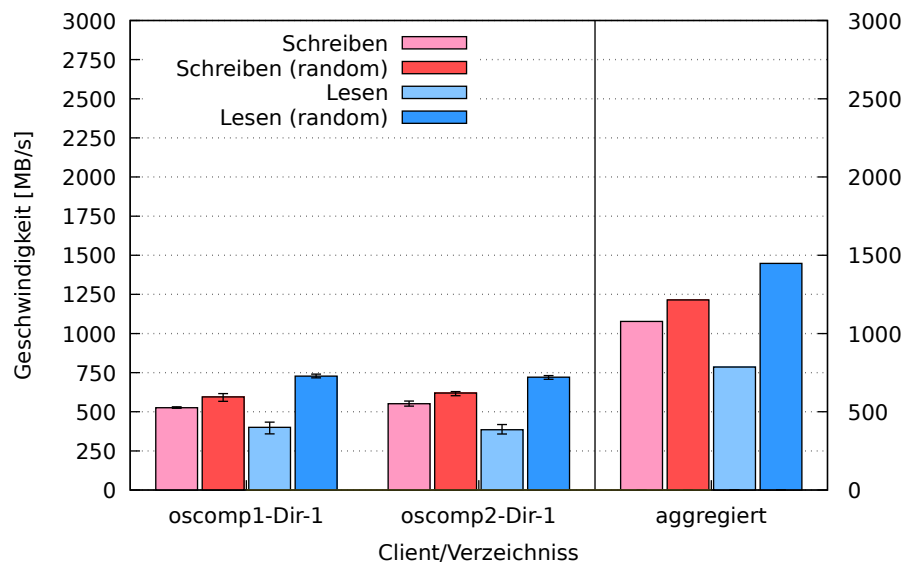


Abbildung 8.9: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

8.1.3.1 Zwischenergebnis

Die Schreibwerte liegen zwischen ca. 625 MB/s (langsamster Wert beim cephinternen Benchmark) und ca. 1250 MB/s (summierter Wert beim Ceph-Filesystem). Da drei Kopien gespeichert werden müssen, bevor es ein Acknowledgement gibt, wird entsprechend viel Zeit gebraucht. Insgesamt kann gesagt werden, dass die Schreibgeschwindigkeit abnimmt, je höher die Replikationszahl ist.

Die Lesewerte bleiben unverändert zum Versuch mit zwei Repliken. Sie scheinen nicht durch die Veränderung der Replikationszahl beeinflusst zu werden, da Ceph nicht mehrere Repliken gleichzeitig zu lesen scheint. Wird also bspw. eine Datei gelesen, die aus 100 Objekten besteht und 3-fach repliziert ist, werden nicht 300 Objekte, sondern nur 100 gelesen. Die Anzahl der Repliken spielt für das Lesen also keine Rolle.

8.1.4 Vergleich zur Konfiguration ohne SSDs

Der durchgeführte Test erlaubt einen Vergleich mit der Konfiguration des Ceph-Clusters aus Kapitel 7.2, da hier die gleiche Replikationszahl (nämlich 3) zu Grunde liegt. Der einzige Unterschied liegt darin, dass die Journale bei der jetzigen Konfiguration auf SSDs liegen, während sie beim Cluster in der Standardkonfiguration auf den gleichen Laufwerken wie die OSDs verortet waren. Die ca. doppelt so hohe Schreibleistung ist dadurch zu begründen, dass die SSDs auch ca. doppelt so schnell wie die verbauten Festplatten schreiben und die

Festplattenschreibköpfe nicht zwischen Journalen und Datenblöcken hin und her pendeln müssen, da die Journale nun extra auf separaten SSDs liegen.

Die Leseleistung hat sich nicht verändert. Sie ist mit und ohne SSDs gleich schnell. Der Grund liegt darin, dass Ceph die Daten nicht aus den Journalen liest, sondern direkt vom OSD. Ob ein Journal auf einer SSD oder sich drehenden Festplatte liegt, spielt daher für die Leseleistung keine Rolle.

8.2 Bluestore, SSDs, Replikation

Die nächste Stufe der Umkonfiguration des Clusters besteht im Wechsel des Speicherbackends von Filestore zu Bluestore. Wie schon im Kapitel 2.1 beschrieben, verwendet der Bluestore beim Speichern auf einem OSD kein Linux-Dateisystem mehr, sondern schreibt direkt auf das Block-Device. Seine Metadaten werden in einer schnellen RocksDB gespeichert und alle Schreibvorgänge in einem WAL (write-ahead-log) protokolliert, so dass die Metadatenbank nach einem Ausfall wiederhergestellt werden kann. In dieser Konfiguration werden Metadatenbank und WAL auf einer separaten SSD gespeichert, um höchstmögliche Geschwindigkeitsvorteile erzielen zu können. Die Metadatenbank und das WAL können theoretisch auf der gleichen Festplatte wie das OSD liegen. Da jedoch SSDs zur Verfügung stehen und ein Vergleich zum Filestore mit SSDs gezogen werden soll, wurde die Konfiguration mit SSDs verwendet.

Bluestore wird zwar seit Ceph-Version „jewel“ unterstützt, die Installation von Metadatenbanken und WALs auf separaten SSDs ist damit jedoch nur schwer zu realisieren, da die notwendigen Befehle in dieser Version noch nicht vorhanden sind. Erst mit Version „luminous“, das noch nicht ganz fertig entwickelt ist und nur als Release-Candidat vorliegt, lassen sich SSDs und OSDs für Bluestore sauber trennen und konfigurieren. Deshalb wurde für die kommenden Tests die Version „luminous“ von Ceph installiert. Dafür ist es nötig, einen Kernel zu installieren, der mindestens Version 4.11 hat, da sonst „luminous“ und einige Funktionen des RADOS Block-Devices nicht funktionieren [28].

Kernel 4.11 und „luminous“ installiert man folgendermaßen:

Listing 8.3: Installation von Kernel 4.11 und luminous

```

# Installation des neuesten Kernels in Ubuntu 16.04 (weicht ab bei anderen
  Distributionen)
root@osceph6:~# apt-get install linux-image-generic-hwe-16.04-edge

# Public-Key herunterladen, um die Echtheit der Ceph-Pakete sicherstellen zu können
5 root@osceph6:~# wget -q -O- 'https://download.ceph.com/keys/release.asc' | sudo apt-
  key add -

# Eintragen der Ceph-Paketquellen in die entsprechende Datei für apt
root@osceph6:~# echo deb https://download.ceph.com/debian-luminous/ $(lsb_release -sc)
  main | sudo tee /etc/apt/sources.list.d/ceph.list

10 # Update der Paketquellen und Installation von ceph-deploy
root@osceph6:~# sudo apt-get update && sudo apt-get install ceph-deploy

# Ceph in der Version luminous auf den Ceph-Servern installieren
root@osceph6:~# ceph-deploy install --release luminous osceph2 osceph3 osceph4 osceph5
  osceph6

```

Die notwendigen Befehle [9] zum Einrichten von OSDs mit Bluestore und separater Metadatenbank samt WAL lauten dann:

Listing 8.4: Beispiel für Einrichtung zweier Bluestore OSDs samt Metadatenbanken und WALs

```

1 # für OSDs gewünschte Festplatten formatieren
root@osceph6:~# ceph-deploy zap osceph6:sde osceph6:sdf

# OSDs sowie Metadatenbank und WAL vorbereiten (Die Festplatten für das OSD sind /dev/
  sde und /dev/sdf. Die SSD für beide ist /dev/sdc.)
root@osceph6:~# ceph-disk prepare --bluestore /dev/sde --block.wal /dev/sdc --block.db
  /dev/sdc
6 root@osceph6:~# ceph-disk prepare --bluestore /dev/sdf --block.wal /dev/sdc --block.db
  /dev/sdc

# OSDs aktivieren und in den Cluster einfügen. (Die SSD muss nicht angegeben werden.
  Sie ist bereits mit den OSDs assoziiert.)
root@osceph6:~# ceph-disk activate /dev/sdf1 /dev/sdc1

11 # Ggfs. muss vor der Aktivierung die Datei: /var/lib/ceph/osd/ceph-NUMMERDESOSDS/block
  umbenannt und nach der Aktivierung wieder zurückumbenannt werden, da hier ein
  falscher symbolischer Link gesetzt sein kann.

```

Nachdem diese Schritte für alle OSDs und Ceph-Server durchgeführt worden sind, kann mit dem Benchmarking fortgefahren werden. In folgenden Test wird zunächst wieder ein Replikationspool benutzt und drei verschiedene Replikationszahlen durchprobiert.

8.2.1 Replikationszahl 1

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „luminous“ 12.1.4 (Release Candidate)
- Bluestore als Backend
- Pro Ceph-Server zwei SSDs mit je sechs RocksDB-Metadatenbanken und write-ahead-logs
- Poolart Replikation
- Replikationszahl eins
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, ausgeführt um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

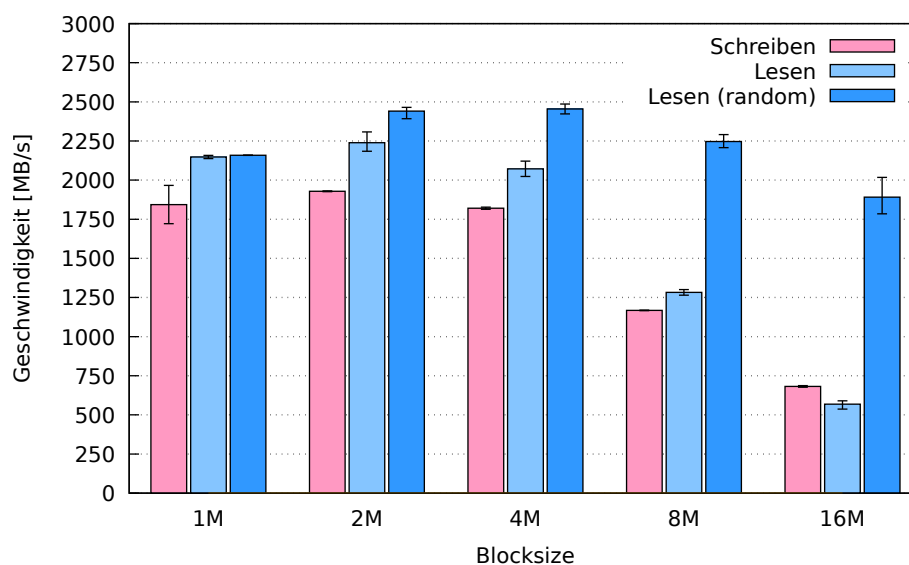


Abbildung 8.10: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

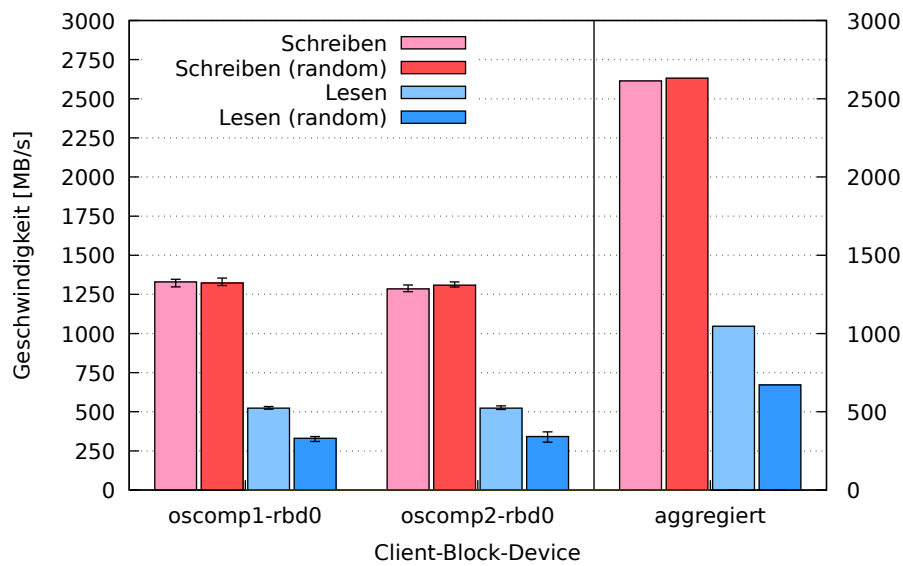


Abbildung 8.11: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

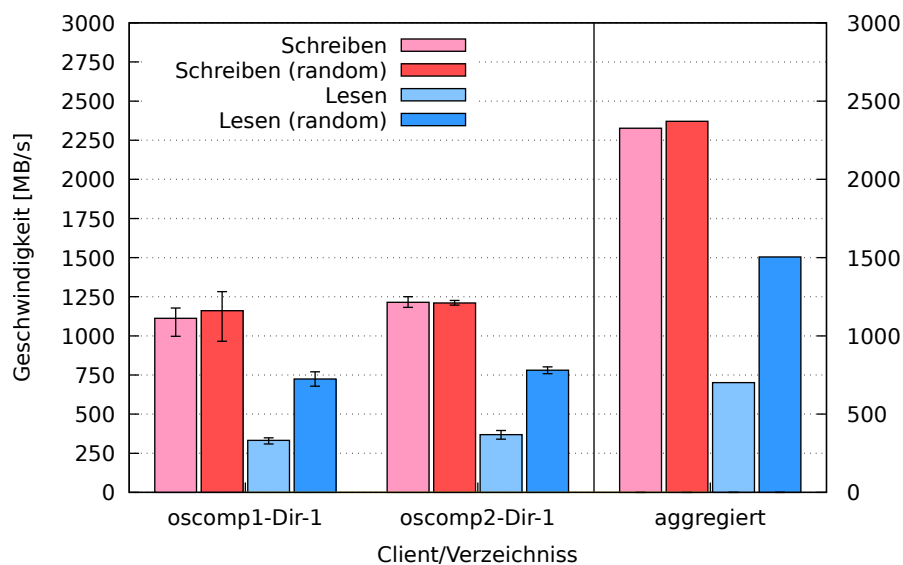


Abbildung 8.12: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

8.2.1.1 Zwischenergebnis

Die Schreibraten zeigen bei allen drei Tests hohe Werte. Es irritiert, dass der ceph-interne Benchmark geringere Schreibwerte anzeigt, als die beiden realistischeren fio-Benchmarks. Bei diesen beiden, dem Block-Device Test und dem Filesystem-Test, werden schließlich „echte“ Dateien vom Client zum Cluster geschickt und dabei die Zeit gestoppt. Als Erklärung für die Diskrepanz zum ceph-internen Benchmark sei auf Kapitel 7.1.4 verwiesen. Die

Messwerte des ceph-internen Benchmarks sind also eher als Orientierung denn absolute Grenzwerte der Clusterleistungen zu verstehen.

Ungeachtet dieses Effektes, sind die Schreibraten hoch und wieder darauf zurückzuführen, dass nur ein Exemplar jeden Objektes gespeichert werden muss, da die Replikationszahl bei eins liegt.

Die Lesewerte sind durch die Bank weg hoch, obwohl dieses Mal der Block-Device Test die geringeren Werte aufweist. Das Ceph-Filesystem zeichnet sich bei diesem Test insbesondere durch die hohen Werte beim wahlfreien Lesen aus. Ein Resultat, dass sich beim cephinternen Benchmark ebenso, beim Block-Device aber nicht zeigt. Die Richtigkeit der gemessenen Werte wurde dadurch verifiziert, dass die Tests mehrfach ausgeführt wurden.

8.2.2 Replikationszahl 2

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „luminous“ 12.1.4 (Release Candidate)
- Bluestore als Backend
- Pro Ceph-Server zwei SSDs mit je sechs RocksDB-Metadatenbanken und write-ahead-logs
- Poolart Replikation
- Replikationszahl zwei
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, ausgeführt um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

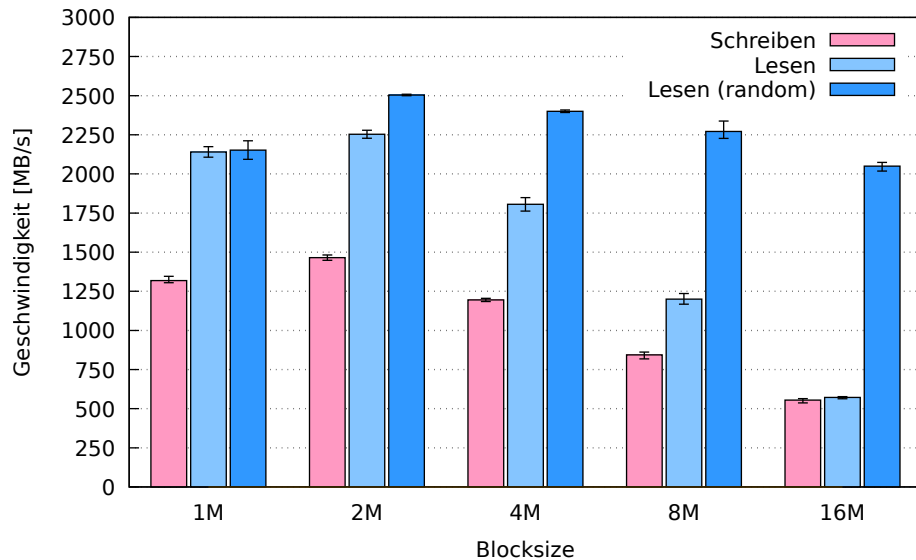


Abbildung 8.13: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

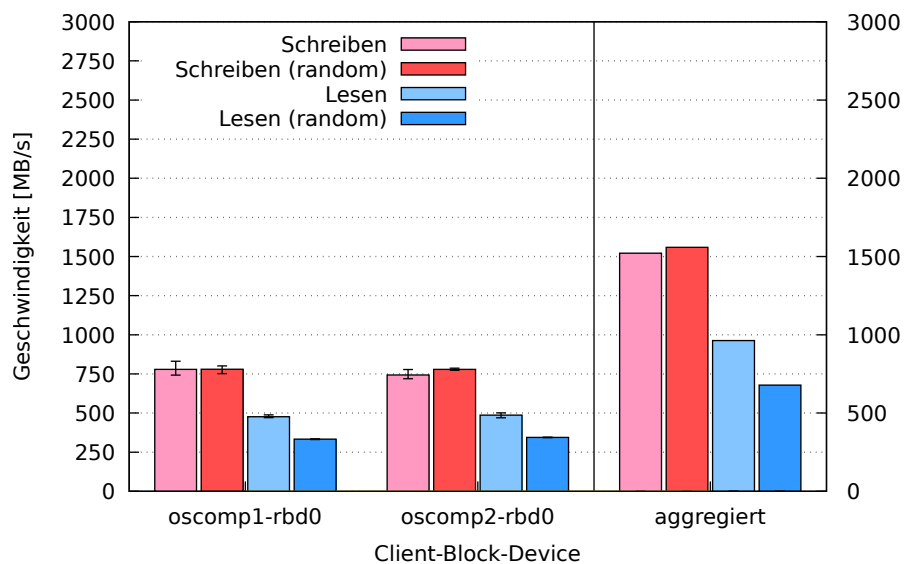


Abbildung 8.14: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

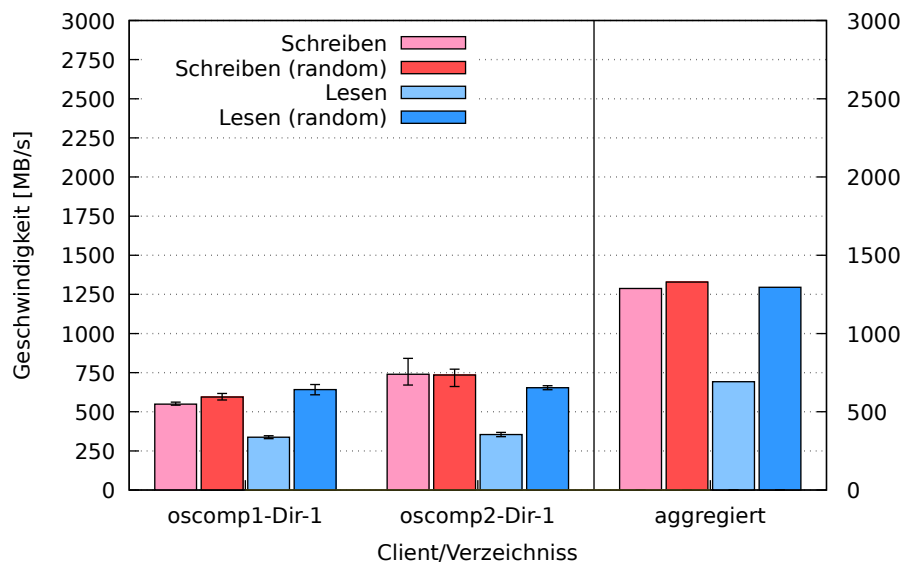


Abbildung 8.15: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

8.2.2.1 Zwischenergebnis

Die Schreibgeschwindigkeit hat erwartungsgemäß bei allen drei Tests im Vergleich zum Test mit der Replikationszahl eins abgenommen, da jetzt zwei Exemplare jeden Datenobjektes gespeichert werden müssen.

Die Lese- und Schreibgeschwindigkeit ist bei allen drei Tests gleich geblieben.

8.2.3 Replikationszahl 3

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „luminous“ 12.1.4 (Release Candidate)
- Bluestore als Backend
- Pro Ceph-Server zwei SSDs mit je sechs RocksDB-Metadatenbanken und write-ahead-logs
- Poolart Replikation
- Replikationszahl drei
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)

- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, ausgeführt um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

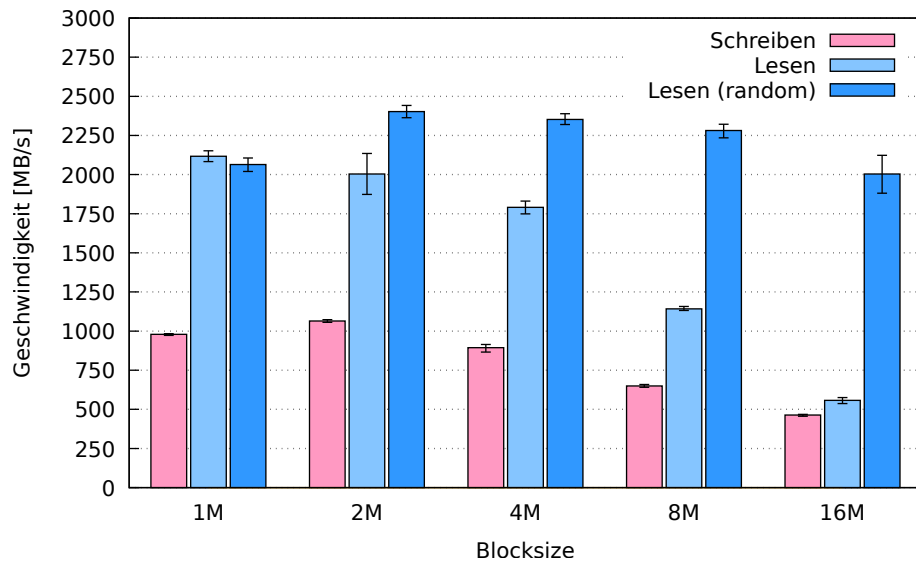


Abbildung 8.16: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

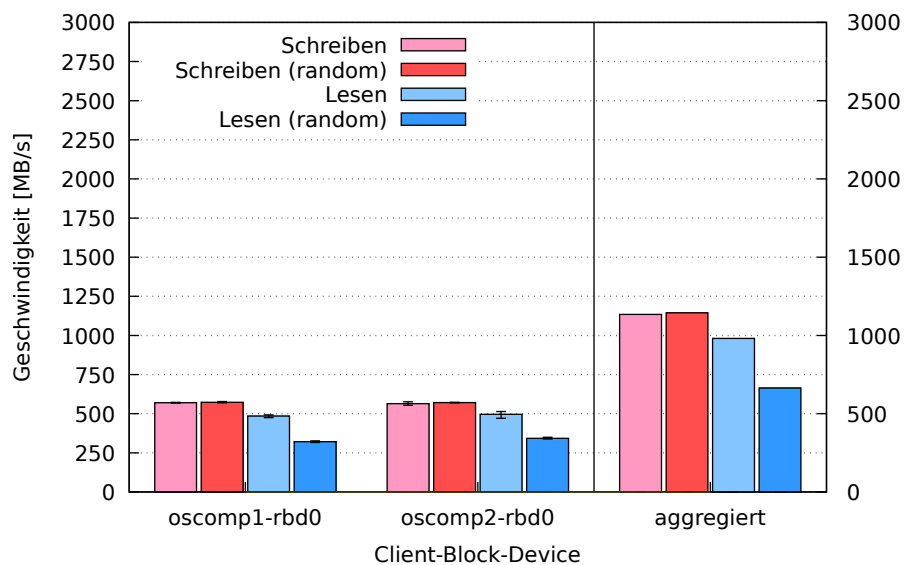


Abbildung 8.17: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

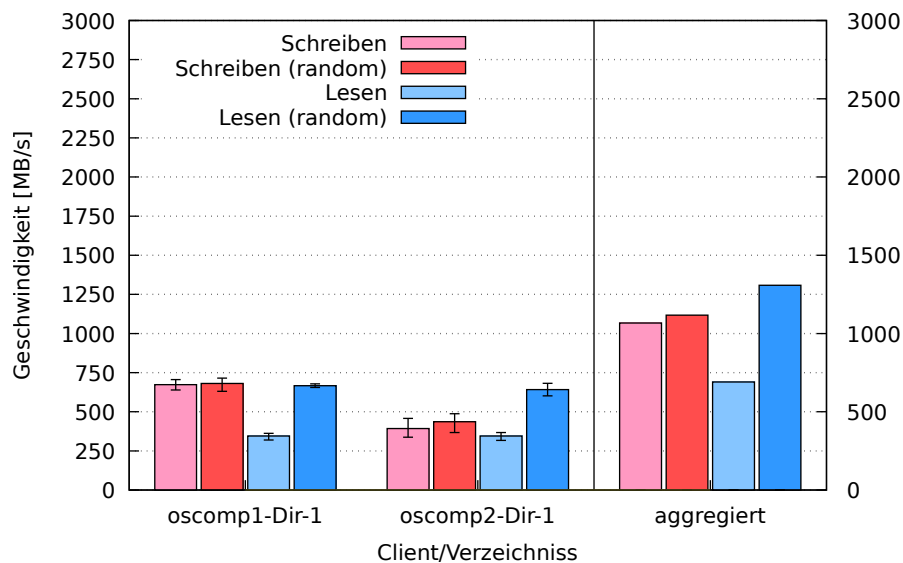


Abbildung 8.18: Schreib-/Leseraten, bei je einem gemounteten Verzeichnis pro Host

8.2.3.1 Zwischenergebnis

Die Schreibgeschwindigkeit ist bei allen drei Tests gesunken, da die Replikationszahl gestiegen ist und damit mehr Kopien gespeichert werden müssen.

Die Lesegeschwindigkeit bleibt unverändert.

8.2.4 Vergleich zum Filestore mit SSDs

Im Vergleich zum Filestore fällt auf, dass beim Bluestore die Schreibleistung für kleine Blocksizes beim cephinternen Benchmark höher ausfällt, dagegen bei größeren Blocksizes geringer. Bei den Leseleistungen zeigen sich allerdings dramatische Unterschiede. Der Filestore ist bei allen Replikationszahlen schneller als der modernere Bluestore. Eine Erklärung dafür kann das jüngere Alter des Bluestores sein, der noch am Anfang seiner Entwicklung steht. Sage Weil schreibt, dass es „noch viel mehr zu tunen und tweaken“ [2] gibt. Ein weiterer Grund könnte darin liegen, dass die Ceph-Version „luminous“ noch nicht final ist, sondern nur als Release Candidate vorliegt.

8.3 Bluestore, SSDs, Erasure-Coding

Nachdem im vorigen Abschnitt der Bluestore mit einem Replikationspool getestet worden ist, soll nun ein Pool verwendet werden, der mit Erasure-Coding (siehe Abschnitt 2.1)

funktioniert. Beim Erasure-Coding werden keine ganzen Kopien von Datenobjekten gespeichert, wie bei der Replikation, sondern zu einer Sequenz von Daten-Chunks eine bestimmte Anzahl Coding-Chunks. Jedes Objekt wird in „m“ Daten-Chunks und „k“ Coding-Chunks geteilt. Es müssen $k+1$ Chunks ausfallen, bevor ein Datenobjekt nicht mehr rekonstruiert werden kann. Im Gegensatz zur Replikation spart man durch diesen Ansatz Speicherplatz, da nicht mehr ganze Kopien eines Objektes gespeichert werden müssen, um Redundanz zu erzeugen. Erkauft wird dieser Vorteil durch höhere CPU-Last beim Erzeugen und Verarbeiten der Chunks, die im Hauptprozessor errechnet werden müssen. Zum Verständnis wird oft der Vergleich zum RAID herangezogen. Ein Erasure-Coding mit einer Anzahl an Coding-Chunks von eins, entspräche einem RAID 5. Bei zwei Coding-Chunks hätte man ein RAID 6. Da man die Anzahl der Coding-Chunks beliebig steigern kann, wären quasi RAIDs mit beliebiger Parität möglich, nur mit dem Unterschied, dass sich das RAID nicht nur über einen einzelnen Rechner, sondern einen Verbund erstreckt.

Es ist zu erwähnen, dass die Konfiguration des Erasure-Codings mit dem RADOS Block-Device nur auf Bluestore-Backends möglich ist und dies auch nur seit der Version „luminous“, weshalb sie auch bei diesem Test wieder zugrunde liegt [29]. Die Nutzung des Ceph-Filesystems mit Bluestore und Erasure-Coding ist ganz fehlgeschlagen. Es wird noch nicht vollständig unterstützt [30]. Versucht man dennoch einen Erasure-Coding Pool für Ceph-Filesystemoperationen zu verwenden, versagt der Cluster seinen Dienst und gerät in einen Fehlerzustand, der nur durch das Löschen des Pools wieder zu bereinigen ist.

Für das RADOS Block-Device und den cephinternen Benchmark konnte das Erasure-Coding allerdings eingesetzt werden. Im Folgenden sei beschrieben, wie man Erasure-Coding für das RADOS Block-Device nutzt.

Zunächst installiert man einen Ceph-Cluster mit Bluestore wie in Abschnitt 8.2 beschrieben, dann erstellt man ein Erasure-Coding Profile, in dem man die Anzahl der Daten- und Coding-Chunks festlegt. Danach wird ein Datenpool mit eben diesem Profile und ein extra Metadatenpool erstellt, der nach wie vor ein Replikationspool ist und beim Mounten auf dem Client als Quelle für das Image angegeben wird. Wichtig ist hier nach wie vor, dass der Linux Kernel in Version 4.11 (4.5 auf dem Server und 4.11 auf dem Client) oder höher vorliegen muss, damit das Block-Device verwendet werden kann [31].

Listing 8.5: Erzeugen eines Erasure-Coding Pools für das RADOS Block-Device

```
# Erzeugen eines Erasure-Coding Profils EC-profile mit zwei Daten- und einem Coding
  Chunk
root@osceph6:~# ceph osd erasure-code-profile set EC-profile ruleset-failure-domain=
  host k=2 m=1

4 # Erzeugen eines Pools namens EC-pool verknüpft mit dem erzeugten EC-profile und 1024
  PGs
root@osceph6:~# ceph osd pool create EC-pool 1024 1024 erasure EC-profile

# Erlauben partieller Overwrites für den EC-Pool.
root@osceph6:~# ceph osd pool set EC-pool allow_ec_overwrites true
```

8.3.1 Anzahl Coding-Chunks 1

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „luminous“ 12.1.4 (Release Candidate)
- Bluestore als Backend
- Pro Ceph-Server zwei SSDs mit je sechs RocksDB-Metadatenbanken und write-ahead-logs
- Poolart Erasure-Coding (Datenpool: Erasure-Coding, Metadatenpool: Replikation mit Replikationszahl 3)
- Datenpool: 2 Data Chunks und 1 Coding Chunk
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

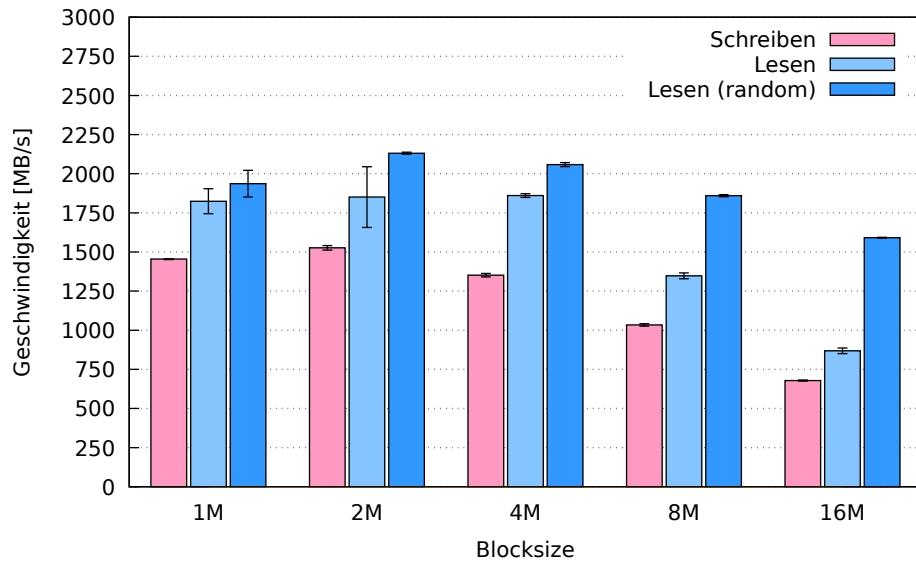


Abbildung 8.19: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

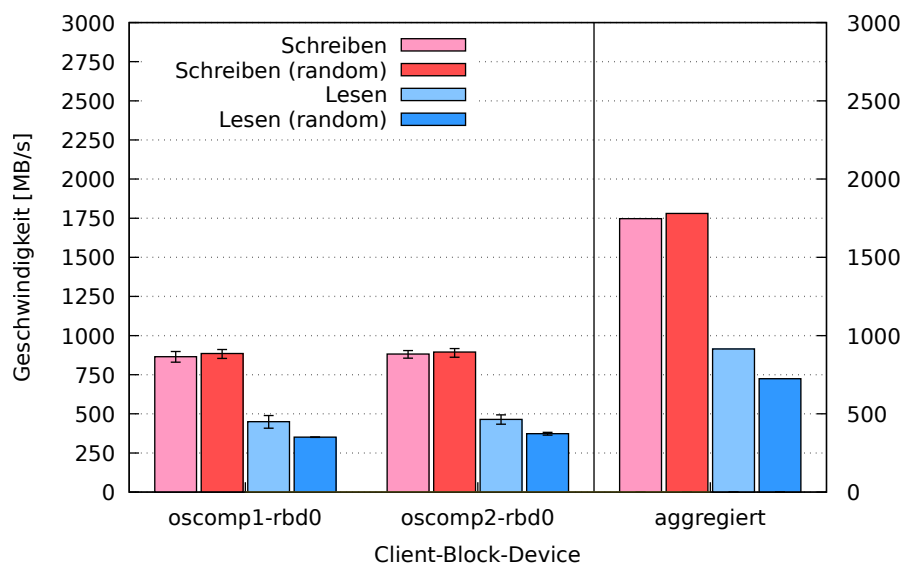


Abbildung 8.20: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

8.3.1.1 Zwischenergebnis

Die Schreibleistung bei einem Coding Chunk pendelt zwischen ca. 720 MB/s und 1.500 MB/s beim cephinternen Benchmark und 1.750 MB/s beim RADOS Block-Device und

liegt damit ähnlich hoch wie beim vergleichbaren Test des Replikationspools mit Bluestore aus dem vorherigen Test.

Die Leseleistung fällt dagegen unterschiedlich aus. Beim wahlfreien Lesen hängt der Replikationspool mit Bluestore den Erasure-Coding Pool klar ab. Beim sequenziellen Lesen ist der Replikationspool nur bei kleinen Blockgrößen schneller, bei größeren Blockgrößen zeigt der Erasure-Coding Pool bessere Ergebnisse.

8.3.2 Anzahl Coding-Chunks 2

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „luminous“ 12.1.4 (Release Candidate)
- Bluestore als Backend
- Pro Ceph-Server zwei SSDs mit je sechs RocksDB-Metadatenbanken und write-ahead-logs
- Poolart Erasure-Coding (Datenpool: Erasure-Coding, Metadatenpool: Replikation mit Replikationszahl 3)
- Datenpool: 2 Data Chunks und 2 Coding-Chunks
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

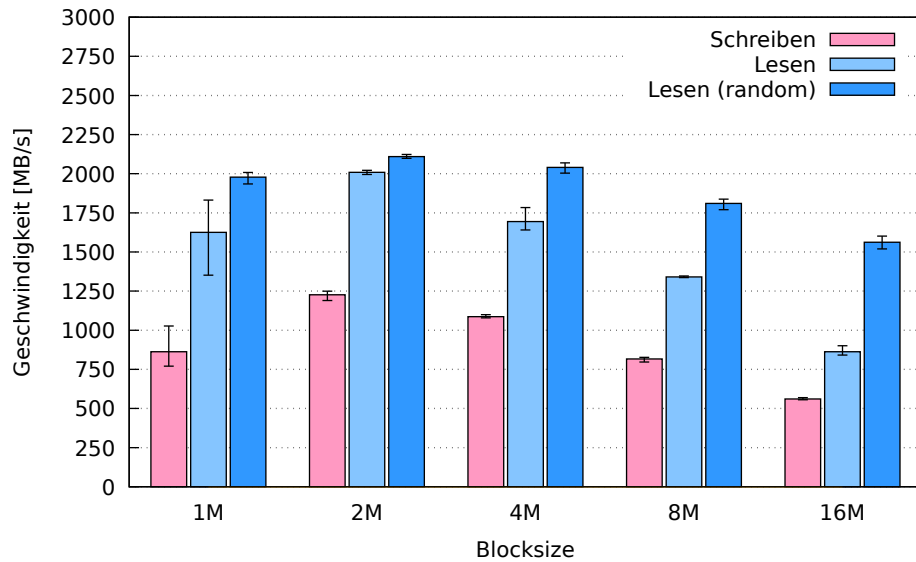


Abbildung 8.21: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

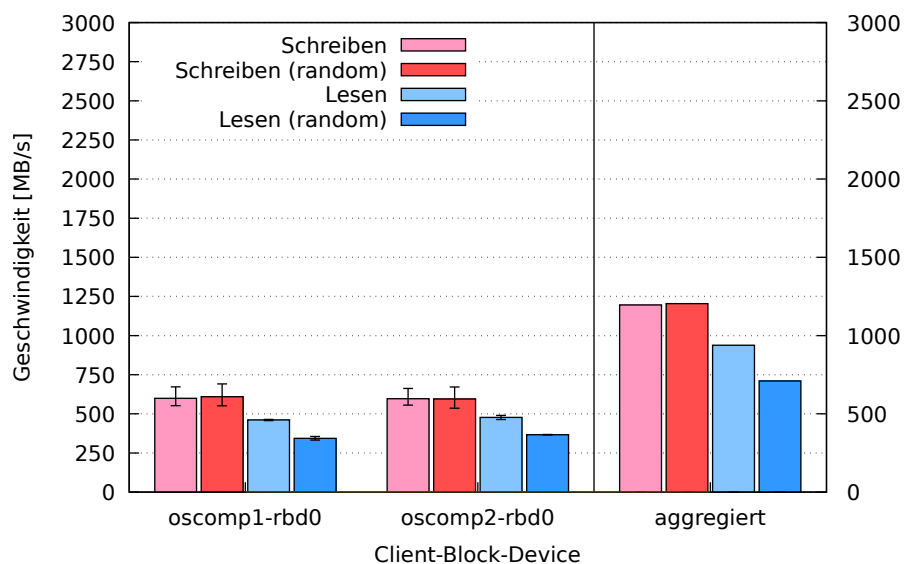


Abbildung 8.22: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

8.3.2.1 Zwischenergebnis

Die Schreibleistung bei zwei Coding-Chunks pendelt zwischen ca. 550 MB/s und 1.250 MB/s beim cephinternen Benchmark und 1.250 MB/s beim RADOS Block-Device und liegt damit höher als beim vergleichbaren Test des Replikationspools mit Bluestore aus

dem vorherigen Test und niedriger als bei dieser Testkonfiguration mit nur einem Coding Chunk.

Die Leseleistung beim cephinternen Benchmark und Block-Device hat sich mit der Erhöhung der Coding-Chunks nicht verändert. Im Vergleich zur vorigen Konfiguration mit Bluestore und Replikationspools, zeigt sich wieder das Bild, dass das wahlfreie Lesen generell langsamer, das sequenzielle Lesen dagegen bei größeren Blöcken mit Erasure-Coding schneller ist.

8.3.3 Anzahl Coding-Chunks 3

Folgende Konfiguration liegt nunmehr auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „luminous“ 12.1.4 (Release Candidate)
- Bluestore als Backend
- Pro Ceph-Server zwei SSDs mit je sechs RocksDB-Metadatenbanken und write-ahead-logs
- Poolart Erasure-Coding (Datenpool: Erasure-Coding, Metadatenpool: Replikation mit Replikationszahl 3)
- Datenpool: 2 Data Chunks und 3 Coding-Chunks
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

Ceph-interner Benchmark - Zunächst wird der cephinterne Benchmark ausgeführt, um die generelle Schreib-/Leseleistung des Clusters unter der gegebenen Konfiguration zu testen.

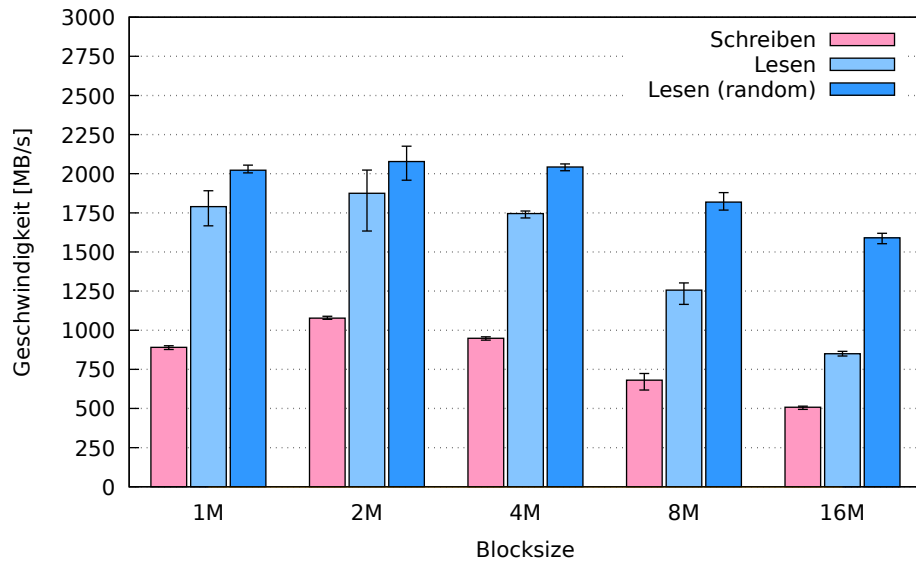


Abbildung 8.23: Schreib-/Leseraten, verschiedene Blocksize, 64 Threads

RADOS Block-Device - Als nächstes folgt der Test des RADOS Block-Devices.

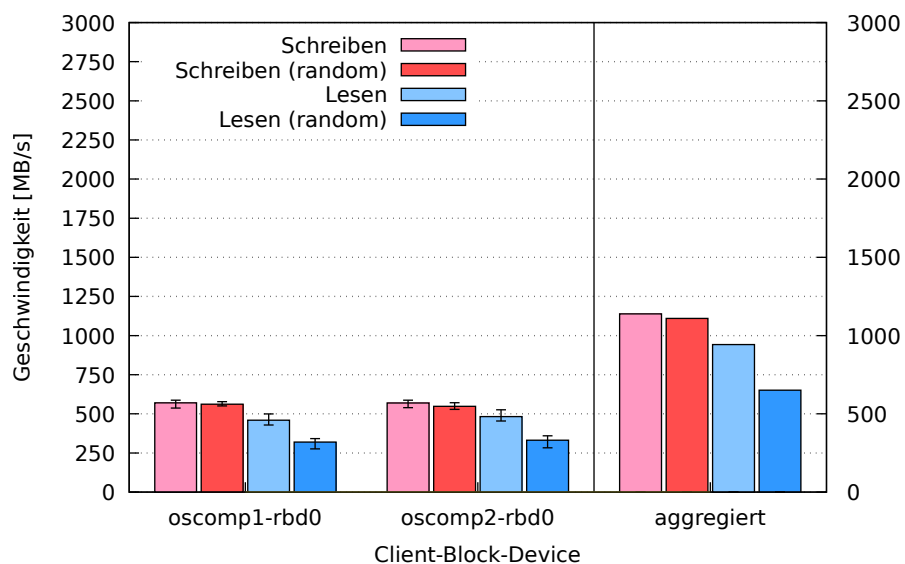


Abbildung 8.24: Schreib-/Leseraten, bei je einem gemounteten Block-Device pro Host

8.3.3.1 Zwischenergebnis

Die Schreibleistung bei drei Coding-Chunks pendelt zwischen ca. 500 MB/s und 1.100 MB/s beim cephinternen Benchmark und 1.250 MB/s beim RADOS Block-Device und liegt damit nur leicht unter den Werten, die bei zwei Coding-Chunks erreicht werden.

Auf die Leseleistung hatte die Erhöhung der Anzahl der Coding-Chunks keine Auswirkung.

8.3.4 Vergleich zum Filestore mit SSDs

Im Vergleich zum Filestore mit SSDs und Replikationspools fällt auf, dass der Erasure-Coding Pool bei vergleichbarer Redundanz deutlich schlechtere Lesewerte erreicht, beim Schreiben aber ähnliche Leistungen erzielt. Die schlechtere Leseleistung liegt an der Anzahl der Daten-Chunks. Sie liegt bei diesem Test bei lediglich zwei. Ceph liest bei dieser Konfiguration nur von zwei OSDs gleichzeitig, wohingegen es beim Replikationspool mit drei Repliken von drei OSDs gleichzeitig lesen kann. Erhöhte man also die Anzahl der Daten-Chunks, würde sich auch die Leseleistung erhöhen.

8.3.5 Vergleich zum Bluestore mit SSDs

Beim Vergleich zum Bluestore mit Replikation fällt auf, dass der Erasure-Coding Pool durch die Bank (etwas) schneller schreibt. Der Grund dafür liegt darin, dass ganz einfach weniger Daten geschrieben werden müssen, da die Chunks ja nur Teile eines Objektes sind, das bei Replikationspools immer vollständig geschrieben werden muss und bei Erasure-Coding Pools eben nur teilweise. Je nach Höhe der Redundanz bzw. Parität in Form von Coding-Chunks verlangsamt sich die Schreibgeschwindigkeit jedoch. Zusammenfassend kann gesagt werden, dass Erasure-Coding Pools schlechtere Leseleistungen bieten, dafür aber schneller schreiben.

Kapitel 9

Betrachtung von Lastszenarien

An ein Speichersystem wie Ceph kann es verschiedene Ansprüche geben, die sich aus spezifischen Anwendungsfällen ergeben können. Diese Ansprüche können etwa Datensicherheit, Geschwindigkeit bei der Datenübertragung oder das Bewältigen vieler gleichzeitiger Zugriffe sein.

Zunächst soll gezeigt werden, wie sich Ceph verhält, wenn es von mehreren Clients und Prozessen aus gleichzeitig benutzt wird (Szenarium: konkurrierende Zugriffe).

Das zweite Szenarium gilt dem Fall der Datensicherheit. Für besonders wichtige Daten ist es sinnvoll, eine hohe Redundanz zu haben, damit sie nach dem Ausfall einer oder mehrerer Komponenten, wie bspw. Festplatten oder ganzer Server, noch verfügbar sind (Szenarium: Datensicherheit).

Das letzte betrachtete Szenarium besteht darin, besonders hohe Geschwindigkeiten beim Lesen und Schreiben von Daten zu erzielen und die Datensicherheit zu vernachlässigen, etwa, wenn es um weniger wichtige Daten, wie Zwischenergebnisse von Rechnungen, geht, die jederzeit wieder reproduziert werden können (Szenarium: Geschwindigkeit).

Im Folgenden sei Ceph hinsichtlich dieser Szenarien begutachtet.

9.1 Konkurrierende Zugriffe

Ceph ist ein verteiltes System, das von mehreren Clients aus und vielen Prozessen gleichzeitig für das Lesen und Schreiben von Daten genutzt werden können muss. Wie sich die Schreib-/ Leseleistung des gesamten Clusters verhält, wenn mehrere Prozesse und Clients parallel auf ihn zugreifen, sei hier dargestellt. Es wurden sowohl RADOS Block-Devices als auch das Ceph-Filesystem getestet. Einmal soll die Leistung bei 8, ein weiteres Mal bei 20 parallelen Zugriffen getestet werden. Dazu wurden zu den beiden Clients „oscomp1“

und „oscomp2“ noch zwei weitere Clients an den Cluster angebunden: „oscomp3“ und „oscomp4“. Sie haben die gleichen Leistungsmerkmale und die gleiche Installation von Ceph wie die beiden schon bekannten Clients.

Die Testkonfiguration sieht wie folgt aus:

- 4 Clients: oscomp1, oscomp2, oscomp3, oscomp4
- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Pro Ceph-Server zwei SSDs mit je 6 Journalen
- Poolart Replikation
- Replikationszahl drei

9.1.1 8 parallele Zugriffe

RADOS Block-Device - Als erstes folgt der Test des RADOS Block-Devices.

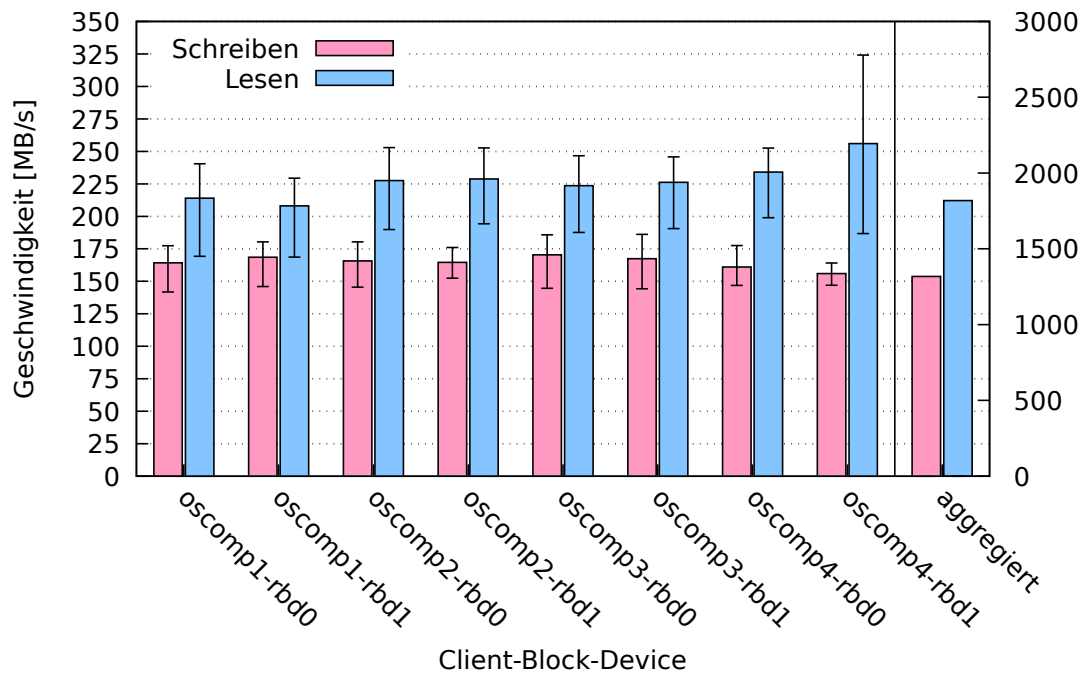


Abbildung 9.1: Schreib-/Leseraten, bei je 2 gemounteten Block-Devices pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

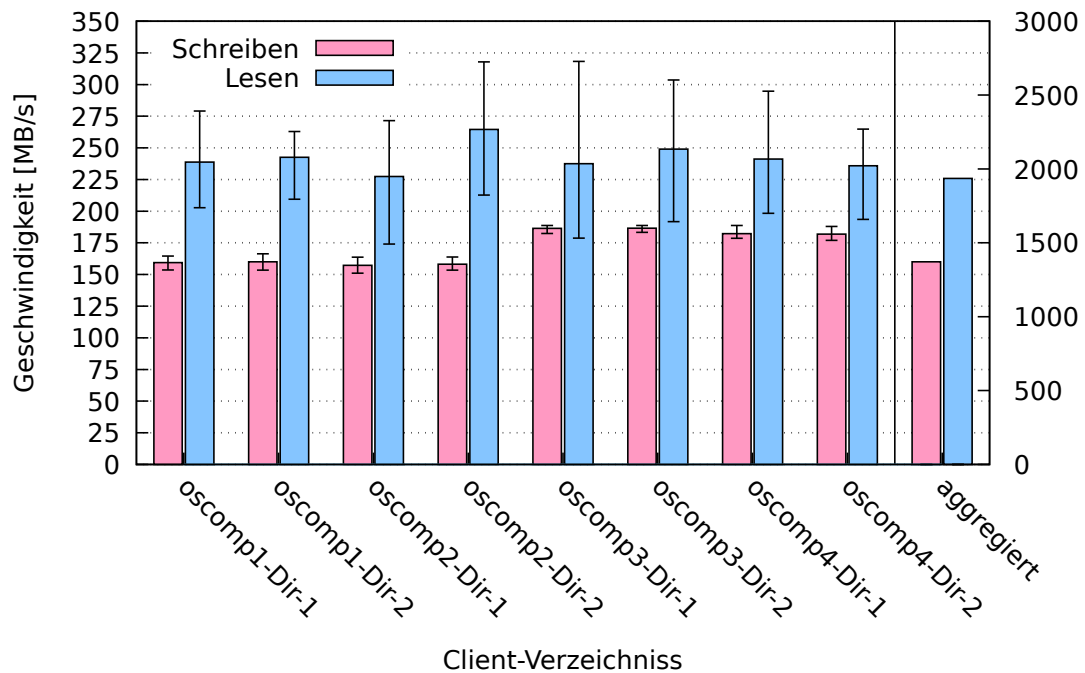


Abbildung 9.2: Schreib-/Leseraten, bei je 2 gemounteten Verzeichnissen pro Host

9.1.2 20 parallele Zugriffe

RADOS Block-Device - Als erstes folgt der Test des RADOS Block-Devices.

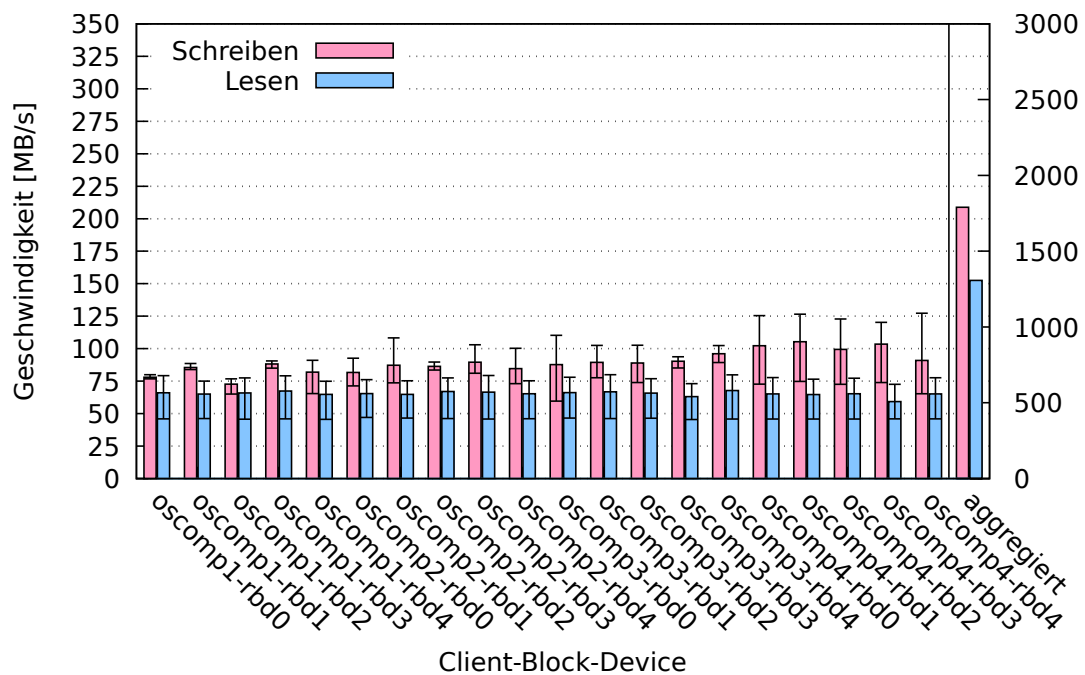


Abbildung 9.3: Schreib-/Leseraten, bei je 5 gemounteten Block-Devices pro Host

Ceph-Filesystem - Der Test des Ceph-Filesystems fiel wie folgt aus:

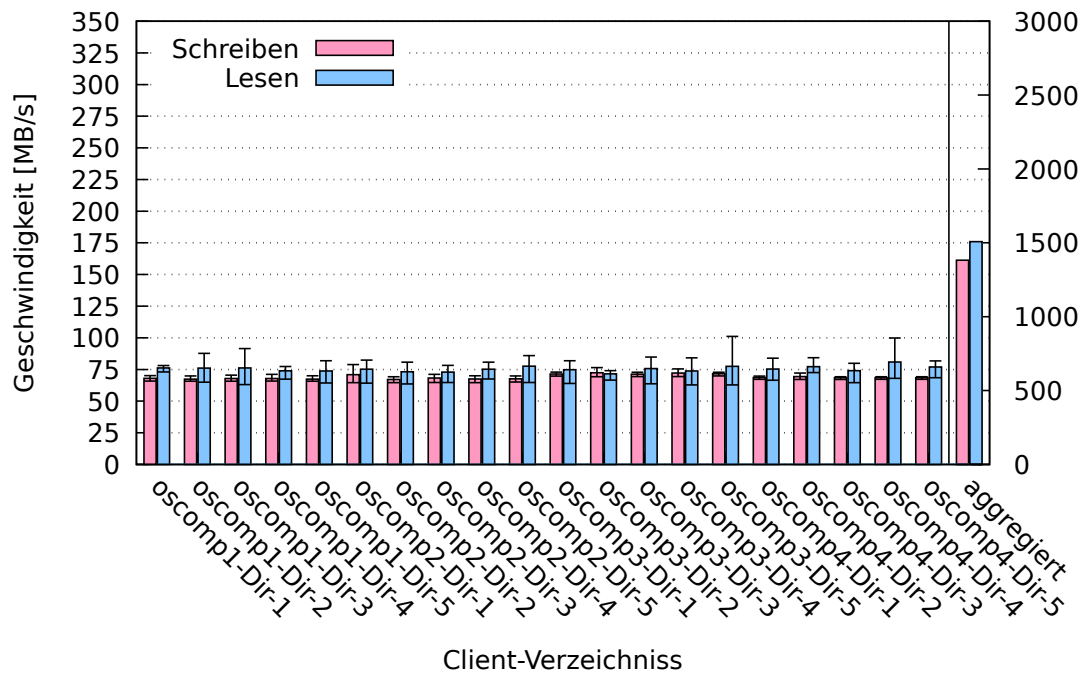


Abbildung 9.4: Schreib-/Leseraten, bei je 5 gemounteten Verzeichnissen pro Host

9.1.3 Auswertung

Es zeigt sich, dass die Schreib-/ Leseleistung des Clusters von 8 parallelen Zugriffen relativ gut ausgereizt wird. Addiert man alle einzelnen Ergebnisse, ergeben sich ungefähr die Daten aus den Messungen des cephinternen Benchmarks in Abschnitt 8.1.3. Viel schneller kann es nicht mehr werden, ohne dass man den Cluster um Ceph-Server erweitert.

Auch mit 20 parallelen Zugriffen kann die Leistung des Clusters nicht mehr erhöht werden. Die einzelnen Prozesse teilen sich vielmehr die Gesamtleistung untereinander auf.

Um diese Gesamtleistung zu optimieren wären zwei Schritte notwendig.

Als erstes müsste man Netzwerkverbesserungen in Betracht ziehen, etwa indem man den Durchsatz der Leitungen erhöht (40 GBit Glasfaserinterfaces), oder indem man den Cluster und seine Clients in unterschiedliche Subnetze sperrt, so dass ein SAN (Storage Area Network) entstünde, in dem die Clientrequests von den Peeringaktionen der Ceph-Server separiert wären und sich nicht gegenseitig die Leitungen streitig machen würden.

Ein weiterer und sehr einfacher Schritt läge in der Vergrößerung des Clusters. Da Ceph, wie noch gezeigt werden wird, beliebig skaliert, lassen sich auch immer mehr Ceph-Server zu einem Cluster hinzufügen und die Schreib-/ Leseleistung beim Zugriff vieler Prozesse gleichzeitig erhöhen. Wie oben beschrieben, sorgt Ceph bzw. CRUSH (siehe Abschnitt 2.1)

dafür, dass die Datenobjekte und ihre Placement-Groups gleichmäßig über den Cluster verteilt werden, so, dass alle Festplatten bzw. OSDs entsprechend ihres Gewichtes (bzw. ihrer Größe) gleichermaßen gefüllt sind. Wenn nun der Cluster größer wäre, dann schreiben und läsen auch mehr Festplatten gleichzeitig an den Objekten einer Datei und die Geschwindigkeit des ganzen Verbundes stiege.

9.2 Datensicherheit versus Geschwindigkeit

Als nächstes soll Ceph in puncto Datensicherheit und Geschwindigkeit evaluiert werden.

Ersteres kann bei Ceph dadurch erreicht werden, dass die Replikationszahl eines Pools erhöht wird, da diese Maßnahme dafür sorgt, dass Datenobjekte mehrfach im Cluster gespeichert werden und bei Ausfall von Ceph-Servern bis zu einem gewissen Grad weiterhin verfügbar sind.

Die Geschwindigkeit des Clusters kann durch verschiedene Maßnahmen gesteigert werden, die allerdings im gewissen Mass zu Lasten der Datensicherheit gehen. Im Allgemeinen erhöht sich die Schreibgeschwindigkeit, wenn man die Replikationszahl senkt, weil dann weniger Kopien eines Objektes gespeichert werden müssen.

9.2.1 Filestore und Bluestore im Vergleich

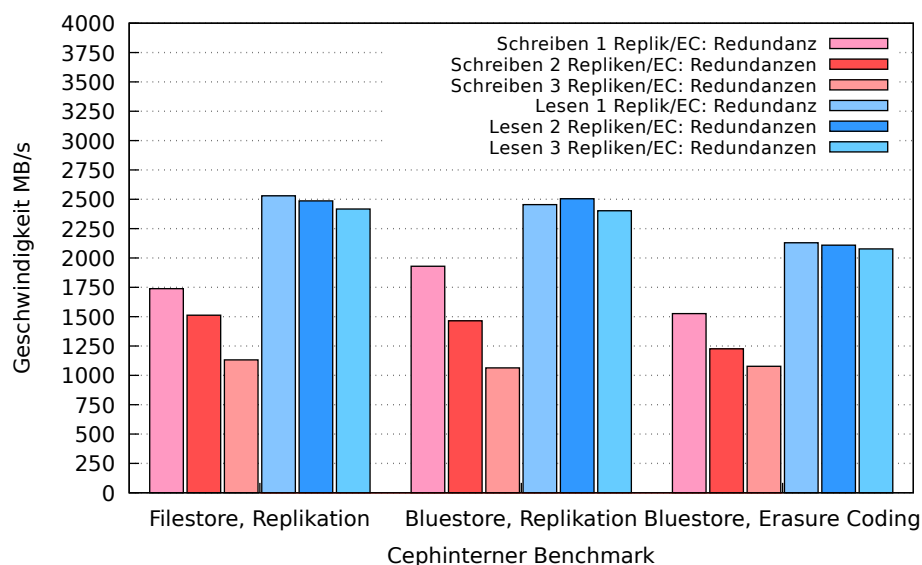


Abbildung 9.5: RADOS-bench im Vergleich der Konfigurationen

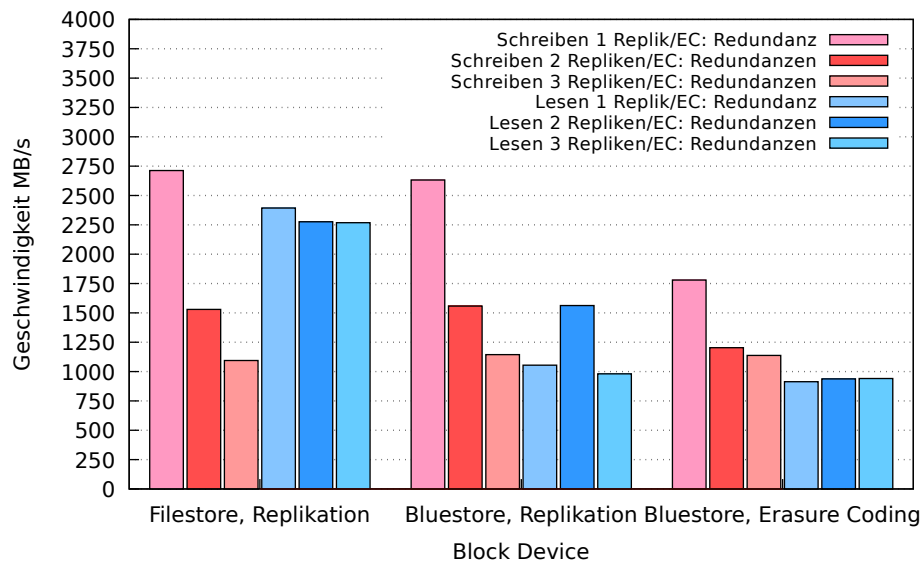


Abbildung 9.6: Block-Device im Vergleich der Konfigurationen

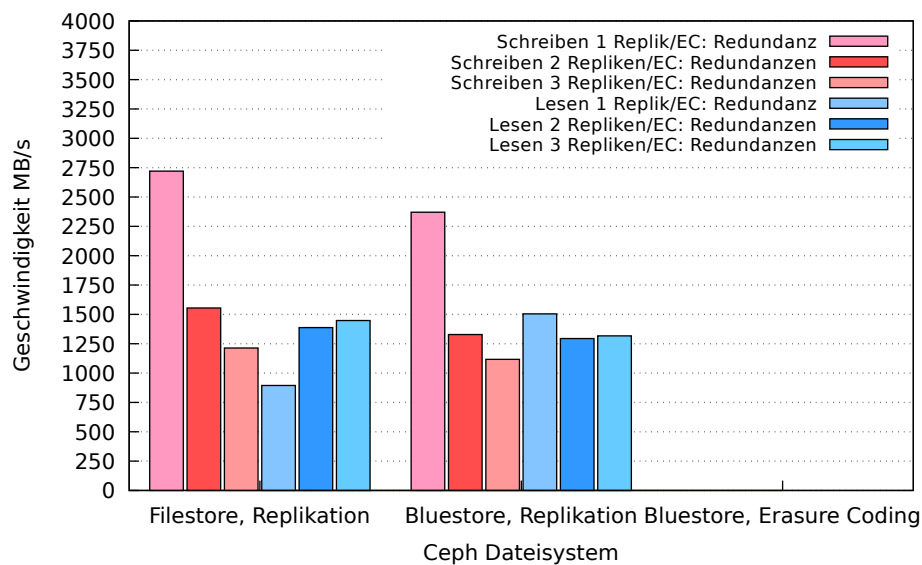


Abbildung 9.7: Ceph-Filesystem im Vergleich der Konfigurationen

9.2.1.1 Interpretation

Die größte Datensicherheit lässt sich in den untersuchten Konfigurationen mit einer Replikationszahl bzw. Coding Chunk Anzahl von drei erreichen.

Die Geschwindigkeit nimmt bei allen drei Testarten mit Zunahme der Replikationszahl und damit der Datensicherheit ab.

Filestore und Bluestore mit Replikationspools unterscheiden sich nicht viel hinsichtlich der Geschwindigkeit. Im synthetischen cephinternen Benchmark weist der Filestore Vorteile beim Lesen auf. Bei den realistischeren Block-Device- und Dateisystemtests ergibt sich

dieser Vorteil allerdings nicht. Das Erasure-Coding zeigt sich im Block-Device Test als leicht unterlegen, deutliche Abstriche gibt es aber auch hier nicht. Da für das Ceph-Filesystem kein Erasure-Coding nutzbar war, können dafür keine Testergebnisse angegeben werden.

Die klare Empfehlung für Datensicherheit und Geschwindigkeit besteht im Filestore mit Replikation, da er vergleichsweise gute Geschwindigkeiten erzielt und außerdem erprobt und schon seit Jahren stabil in Ceph integriert ist, wohingegen Bluestore und Erasure-Coding relative neue Entwicklungen und wenig getestet sind.

Zieht man allerdings den Faktor Brutto-Netto-Speicherplatzverwertung in Betracht, muss die Entscheidung auf den Bluestore mit Erasure-Coding fallen. Er benötigt am wenigsten Speicherplatz, weil, wie oben beschrieben, die Datenobjekte nicht repliziert, sondern geteilt und lediglich mit Paritäten bzw. Coding-Chunks versehen, gespeichert werden, wodurch wesentlich weniger Speicherplatz benötigt wird.

9.2.2 Speicherinterfaces im Vergleich

Im folgenden werden die beiden Ceph-Speicherinterfaces RADOS Block-Device und Ceph-Filesystem hinsichtlich des Verhältnisses zwischen Geschwindigkeit und Datensicherheit verglichen. Der ceph-interne Benchmark wird der Vollständigkeit halber mit einbezogen, obwohl er kein eigenes Ceph-Interface darstellt, sondern nur der Orientierung dient.

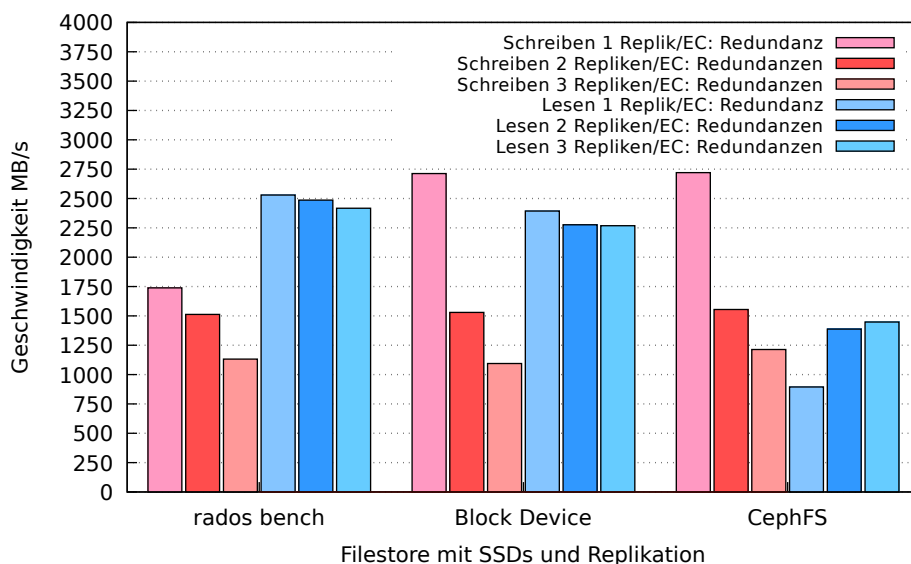


Abbildung 9.8: Vergleich der Tests mit Konfiguration: Filestore, SSDs, Replikation

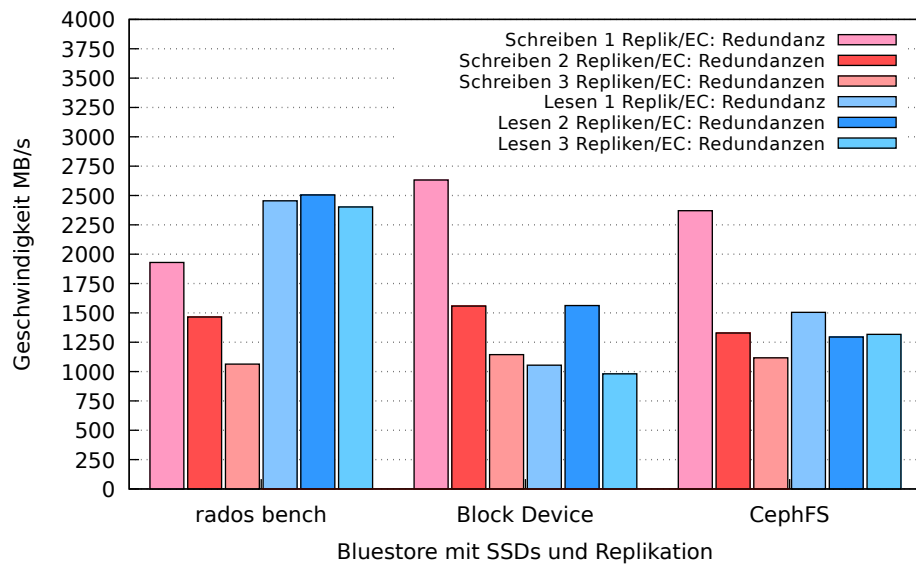


Abbildung 9.9: Vergleich der Tests mit Konfiguration: Bluestore, SSDs, Replikation

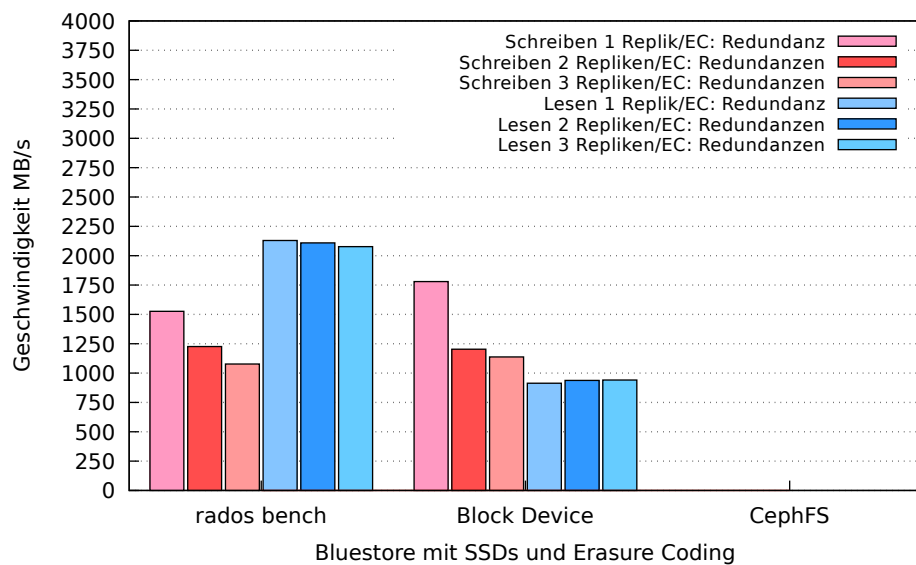


Abbildung 9.10: Vergleich der Tests mit Konfiguration: Bluestore, SSDs, Erasure-Coding

9.2.2.1 Interpretation

Im Vergleich der Ceph-Interfaces performt das Block-Device besser als das Ceph-Filesystem. Es hat vor allem in der Konfiguration mit Filestore und Replikation die wesentlich höheren Lesegeschwindigkeiten und ist auch beim Schreiben leicht im Vorteil.

Des Weiteren muss auch hier angemerkt werden, dass die Zeit, die das Interface bereits in der Entwicklung und im Einsatz ist, eine Rolle für den Gesichtspunkt der Datensicherheit spielt. Das RADOS Block-Device ist schon seit Jahren stabil in Ceph integriert, während das Ceph-Filesystem erst seit Version „jewel“ als stabil betrachtet wird und überdies wich-

tige Eigenschaften dafür noch nicht umgesetzt worden sind. So sind die Metadata-Server des Dateisystems nach wie vor nicht redundant zu betreiben, wie in den Ceph Docs nachgelesen werden kann [32]. Das heißt, mehrere MDS können nicht gleichzeitig laufen und der Ausfall eines einzigen, macht das ganze Dateisystem unzugänglich, was einen Single-Point-of-Failure darstellt. Unter dem Gesichtspunkt der Datensicherheit ist das Ceph-Filesystem zum derzeitigen Zeitpunkt daher nicht zu empfehlen.

Kapitel 10

Auswirkungen von Schadszenarien

Ein großer Vorteil von Ceph liegt darin, dass es sich selbst heilt, wenn eine oder mehrere Komponenten ausfallen. Solch ein Ausfall kann bspw. im Defekt einer oder mehrerer Festplatten oder aber auch dem ungewollten Ausscheiden eines ganzen oder mehrerer Ceph-Server, bspw. durch Netzteil- oder Netzwerkdefekt, liegen. Den Vorgang der Selbstheilung nennt man Recovery.

Wird ein Defekt festgestellt, etwa weil ein oder mehrere OSDs nicht erreicht werden können, wartet Ceph eine festgelegte Dauer (festgelegt in der Variable: „osd down out intervall“ in der Datei „ceph.conf“) und beginnt dann damit, die auf den ausgefallenen OSDs befindlichen PGs auf anderen, durch CRUSH errechneten, OSDs wiederherzustellen. Dieser Vorgang wird solange ausgeführt, bis die eingestellte Replikationszahl wiederhergestellt und eine gleichmäßige Befüllung der OSDs realisiert ist. Während dieses Prozesses ist der Cluster voll funktionstüchtig und Daten können gelesen und geschrieben werden, es sei denn, so viele Komponenten sind ausgefallen, dass eine Wiederherstellung unmöglich ist. Aus diesem Grund sollte die Replikationszahl oder die Anzahl der Coding-Chunks beim Erasure-Coding stets groß genug für die eigenen Datensicherheitsanforderungen gewählt werden.

In diesem Abschnitt wird untersucht, welchen Einfluss das Ausscheiden dreier OSDs oder eines ganzen Ceph-Servers auf die Schreib- und Lesegeschwindigkeit eines Clusters haben. Dabei wird so vorgegangen, dass auf einem Client ein RADOS Block-Device gemountet wird und jeweils 40 GB an Dateien geschrieben und anschließend gelesen werden. Während dieses Prozesses werden ausgewählte Komponenten entfernt und wieder hinzugefügt, wobei jeweils gemessen wird, welchen Einfluss dieser Vorgang auf die Schreib-/ und Leseraten des RADOS Block-Devices hat. Einmal wird dieser Test mit dem Ausfall von drei OSDs

auf einem einzelnen Ceph-Server, ein anderes Mal mit dem simulierten Ausscheiden eines ganzen Ceph-Servers durchgeführt.

Durch folgende Befehle kann der Ausfall von OSDs simuliert und der Recoveryprozess initiiert werden (Beim Ausfall eines ganzen Ceph-Servers werden entsprechend alle OSDs dieses Ceph-Servers ausgeschaltet. Ceph meint dann, den ganzen Server verloren zu haben.):

Listing 10.1: Ausschalten von OSDs und Wiederhinzufügen

```
# Entfernen dreier OSDs
2 root@sceph6:~ > ceph osd out osd.59 osd.58 osd.57

# Wiederhinzufügen der selben OSDs
root@sceph6:~ > ceph osd in osd.59 osd.58 osd.57
```

Folgende Konfiguration liegt auf dem Cluster vor und ist Grundlage der folgenden Tests:

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Pro Ceph-Server zwei SSDs mit je 6 Journalen
- Poolart Replikation
- Replikationszahl drei
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

10.1 Ausfall einzelner OSDs

Zunächst werden drei OSDs ausgeschaltet und der Einfluss auf die Geschwindigkeit eines RADOS Block-Device untersucht. Die Grafiken wurden mit graphite (siehe Kapitel 4.2) aufgezeichnet und mit einer rötlichen Markierung versehen. Sie gibt an, ab welchem Zeitpunkt der Recovery-Prozess einsetzt. Die Zeitintervalle der Grafiken liegen jeweils bei ein bis zwei Minuten. Die y-Achse gibt die Schreib-/ Leseleistung in MB/s an. Die x-Achse beschreibt die Zeit.

10.1.1 Auswirkungen beim Schreiben

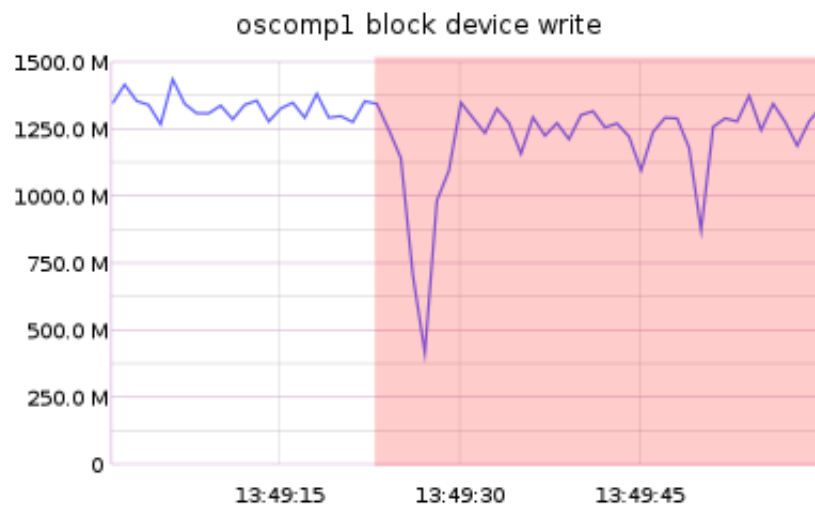


Abbildung 10.1: Veränderung der Schreibleistung bei 3 gleichzeitig ausgefallenen OSDs

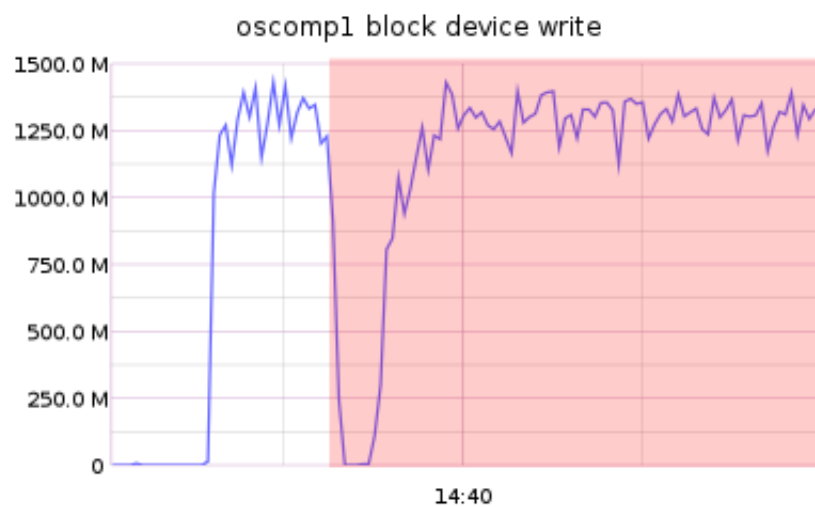


Abbildung 10.2: Veränderung der Schreibleistung bei 3 sich wiedereingliedernden OSDs

Werden die drei OSDs beim Schreiben entfernt, gibt es nur einen kleinen Knick der Kurve und der Schreibprozess wird mit unveränderter Geschwindigkeit fortgesetzt. Das spätere Wiedereingliedern verursacht dagegen einen etwas längeren Ausfall. Das danach einsetzende Schreiben geschieht aber wiederum mit gleichhoher Geschwindigkeit. Der Ausfall dreier OSDs hat also fast keinen Einfluss auf die Schreibleistung des Clusters.

10.1.2 Auswirkungen beim Lesen

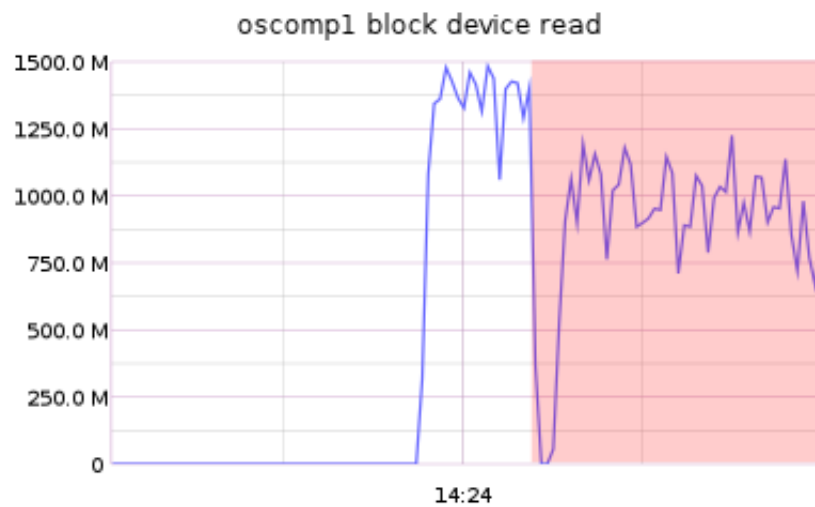


Abbildung 10.3: Veränderung der Leseleistung bei 3 gleichzeitig ausgefallenen OSDs

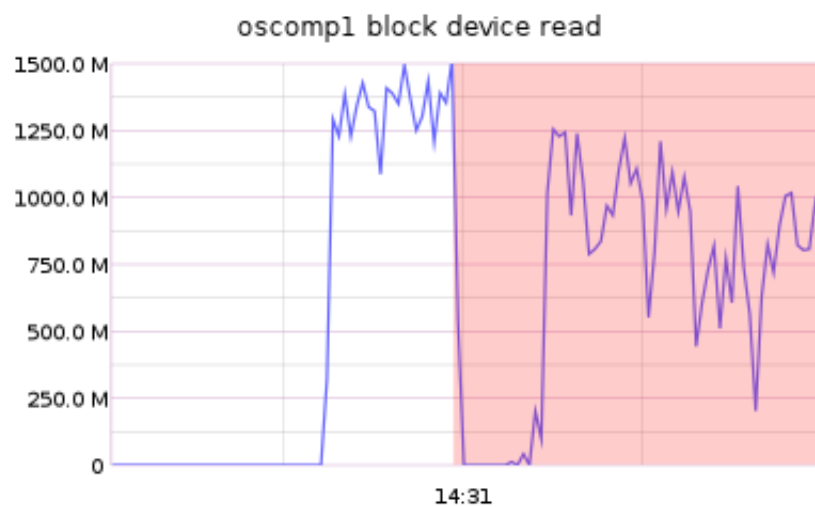


Abbildung 10.4: Veränderung der Leseleistung bei 3 sich wiedereingliedernden OSDs

Das Lesen dagegen verlangsamt sich durch den Ausfall merklich. Es können nicht mehr gleich viele Repliken von Objekten gelesen werden, weil einige ausgefallen sind. In der Zeit, in der sie wiederhergestellt werden, ist das Lesen daher verlangsamt. Auch die Wiedereingliederung wirkt sich negativ auf den Lesevorgang aus. Zudem sind in beiden Fällen kurze Ausfallzeiten zu verzeichnen, in denen gar nicht gelesen wird.

10.2 Ausfall eines Ceph-Servers

Durch das gleichzeitige Entfernen aller OSDs eines Ceph-Servers wird dessen kompletter Ausfall simuliert. Das Wiederanschalten dieser OSDs fingiert den Fall, dass der Server wieder in Betrieb genommen wurde.

10.2.1 Auswirkungen beim Schreiben

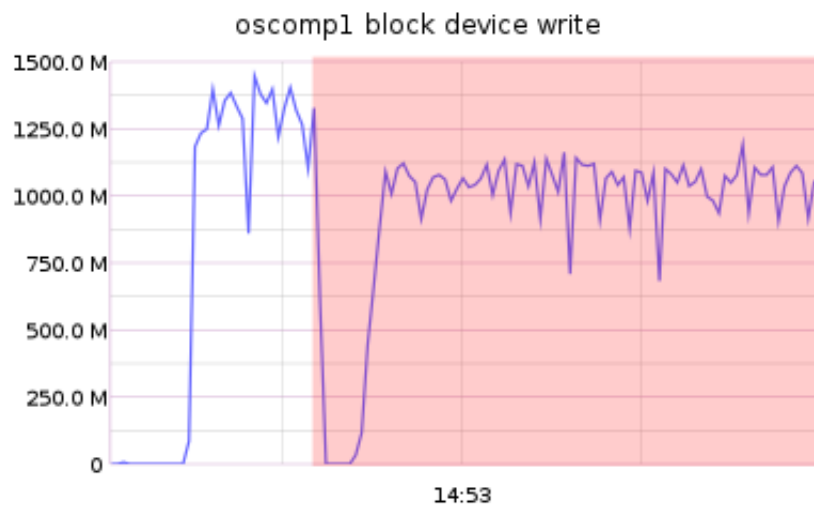


Abbildung 10.5: Veränderung der Schreibleistung bei einem komplett ausgefallenen Ceph-Server

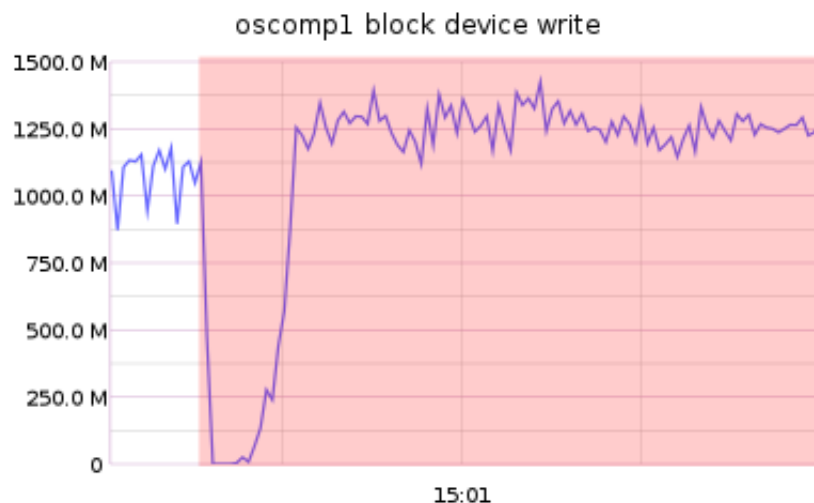


Abbildung 10.6: Veränderung der Schreibleistung bei einem sich wiedereingliedernden Ceph-Server

Beim Ausfall eines ganzen Ceph-Servers, der in diesem Fall 12 von insgesamt 60 OSDs beherbergte, ist eine Reduktion der Schreibleistung während des Recoveryprozesses zu

bemerken. Nach dem Ausfall wird mit niedrigerer Geschwindigkeit geschrieben. Offenbar ist die Menge der verlorenen OSDs entscheidend für die Einschränkung der Schreibleistung beim Recoveryprozess.

Interessant wird es beim späteren Wiedereingliedern. Nach einem kurzen Ausfall sämtlicher Operationen setzt das Schreiben sogar mit erhöhter Geschwindigkeit wieder ein. Der Grund dafür dürfte darin liegen, dass die neu zu schreibenden Daten wieder über alle OSDs verteilt werden können und damit die akkumulierte Schreibleistung des ganzen Clusters zur Verfügung steht, während der Recoveryprozess anscheinend nur wenig Bandbreite verschlingt.

10.2.2 Auswirkungen beim Lesen

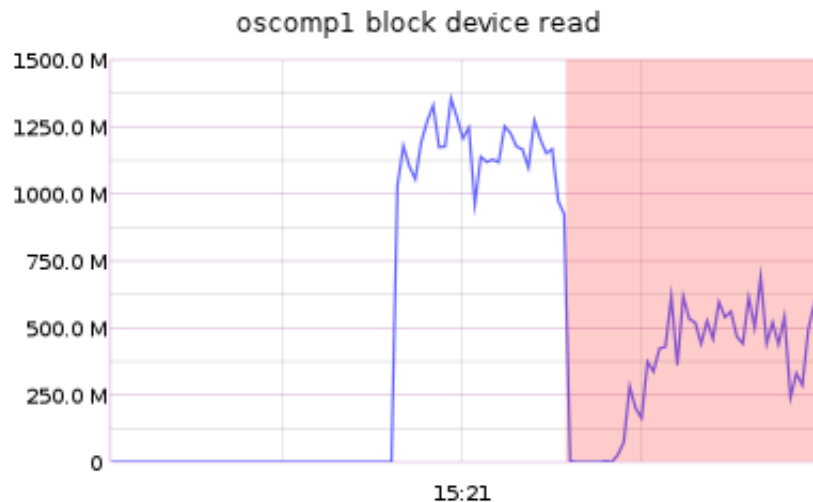


Abbildung 10.7: Veränderung der Leseleistung bei einem komplett ausgefallenen Ceph-Server

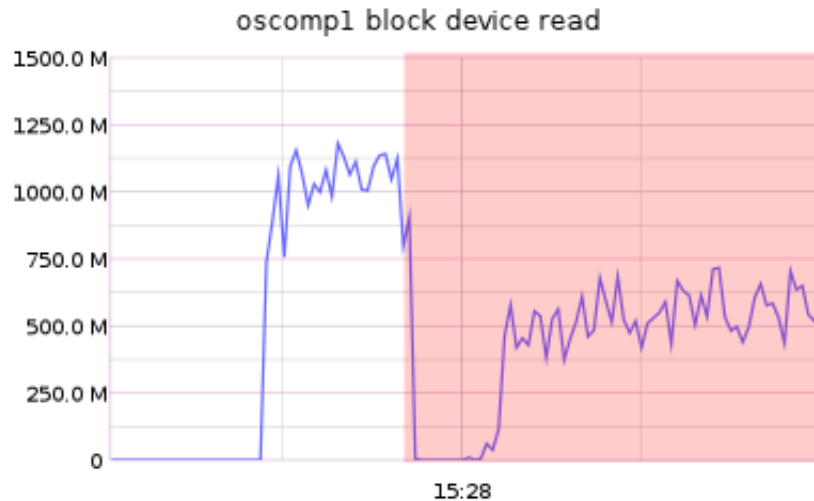


Abbildung 10.8: Veränderung der Leseleistung bei einem sich wiedereingliedernden Ceph-Server

Der Ausfall eines ganzen Servers hat einen verheerenden Einfluss auf die Leseleistung des RADOS Block-Device. Die Kurve sackt beim Eintreten des Recoveryprozesses sehr stark ab. Auch beim Eingliedern liegt die Geschwindigkeit weit unter dem normalen Wert.

10.3 Zusammenfassung

Summa summarum lässt sich sagen, dass die Leseleistung stärker durch den Ausfall von Komponenten beeinflusst wird als die Schreibleistung. Ebenso wirkt sich der nach dem Wiedereingliedern einsetzende Rebalancingprozess auf die Datenraten aus, umso stärker, je mehr Komponenten ausfallen.

Interessanterweise wurde beobachtet, dass der gesamte Recoveryprozess und auch das Rebalancing bei Schreiboperationen bis zu viermal länger dauerte als beim Lesen, was dadurch erklärt werden kann, dass der Cluster beim Lesen ohnehin keine aktive Aufgabe hat und nichts an der Verteilung der Daten im Cluster manipulieren muss, während er beim Schreiben permanent bestrebt ist, den Cluster auszubalancieren und alle OSDs gleichermaßen zu befüllen.

Kapitel 11

Skalierbarkeit

Ceph stellt ein Storage-System dar, das beliebig skalieren kann. Es soll bis in den Exabyte-Bereich Daten verwalten können, ohne an Leistung und Funktionalität einzubüßen. Dabei kann es sowohl vertikal als auch horizontal skalieren. Vertikal bedeutet, dass man in einen vorhandenen Ceph-Server weitere OSDs einbauen kann, ihn also in die Höhe baut. Horizontal meint, es ist möglich, beliebig viele Ceph-Server dem Cluster hinzuzufügen, ihn also in die Breite wachsen zu lassen.

Für die nun erfolgenden Kommandos liegt der Ceph-Cluster in dieser Konfiguration vor:

- Ceph Version „jewel“ 10.2.9 (Major Release)
- Filestore als Backend
- Pro Ceph-Server zwei SSDs mit je 6 Journalen
- Poolart Replikation
- Replikationszahl 3
- insg. 60 Festplatten im Cluster von 5 Ceph-Servern verbaut (siehe Tabelle 5.2 auf Seite 44)
- max. Datendurchsatz des Netzwerkinterfaces eines Clients: rd. 10-20 GBit/s (im Bonding-Mode 4 - 802.3ad)
- max. Schreibleistung aller Festplatten des Clusters zusammen: rd. 10.2 GB/s
- max. Leseleistung aller Festplatten des Clusters zusammen: rd. 10.4 GB/s

11.1 Cluster verkleinern

Um zu zeigen, wie man einen Cluster vergrößert, muss der bestehende zunächst verkleinert werden.

Theoretisch ist es möglich, einen Cluster zu verkleinern, indem OSDs ausgebaut oder Ceph-Server entfernt werden, allerdings tritt dabei bei produktiven Clustern ein Problem auf, für das es nur Workaround-Lösungen gibt. Es ist nicht möglich, die PG-Anzahl eines Pools zu verringern. Dieser Fall ist derzeit in Ceph nicht vorgesehen [33]. Die Workarounds, die dazu im Netz kursieren, sollten auf produktiven Clustern nicht ausgeführt werden, denn sie sind eben nur Ersatzlösungen, ohne Garantie auf Erfolg. Einen Cluster zu verkleinern, geht also nur im begrenzten Maße, solange, wie nicht mehr als 300 PGs pro OSD existieren. Für alles, was darüber liegt, gibt Ceph Health-Warnings aus und gerät in einen Fehlerzustand.

Für den hier vorliegenden Test-Cluster ist das Verkleinern kein Problem, da auf ihm keine mit Daten angefüllten Pools liegen und damit auch deren PG-Anzahl nicht verkleinert werden muss.

Der Clusterzustand verrät, wie viele OSDs eingebaut sind. Man sieht deren Zahl unter „osdmap“.

Listing 11.1: Status des Clusters in voller Ausbaustufe

```
# gibt den Status und die Gesundheit eines Clusters an
root@osceph2:~# ceph -s
cluster 7bd91eb1-8196-41f0-87ca-e94cc3bf60f1
health HEALTH_OK
5 monmap e3: 3 mons at {osceph4=172.30.254.22:6789/0,osceph5=172.30.254.23:6789/0,
    osceph6=172.30.254.24:6789/0}
    election epoch 22, quorum 0,1,2 osceph4,osceph5,osceph6
osdmap e355: 60 osds: 60 up, 60 in
    flags sortbitwise,require_jewel_osds
pgmap v12472: 2048 pgs, 1 pools, 0 bytes data, 1 objects
10 3840 MB used, 218 TB / 218 TB avail
    2048 active+clean
```

Mithilfe der Baumansicht (hier leicht verkürzt dargestellt), kann man sich die so genannte Cluster-Map anzeigen lassen, in der bspw. die Anzahl der Ceph-Server, deren Namen und die Anzahl der OSDs und unter „WEIGHT“ deren Gewicht (nach Größe der Festplatte berechnet, wobei 1TB für einen Wert von 1 steht) zu sehen ist. Vor der Verkleinerung besteht der Cluster aus 60 OSDs, gleichmäßig verstreut über fünf Ceph-Server.

Listing 11.2: Baumstruktur des Clusters in voller Ausbaustufe

```

# gibt die Baumstruktur eines Clusters an
root@osceph2:~/# ceph osd tree
ID WEIGHT  TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
4  -1 218.21320 root default
  -2 43.64264  host osceph2
    ...
  -3 43.64264  host osceph3
    ...
9  -4 43.64264  host osceph4
    ...
  -5 43.64264  host osceph5
    ...
  -6 43.64264  host osceph6
14 48 3.63689    osd.48      up 1.00000      1.00000
    49 3.63689    osd.49      up 1.00000      1.00000
    50 3.63689    osd.50      up 1.00000      1.00000
    51 3.63689    osd.51      up 1.00000      1.00000
    52 3.63689    osd.52      up 1.00000      1.00000
19 53 3.63689    osd.53      up 1.00000      1.00000
    54 3.63689    osd.54      up 1.00000      1.00000
    55 3.63689    osd.55      up 1.00000      1.00000
    56 3.63689    osd.56      up 1.00000      1.00000
    57 3.63689    osd.57      up 1.00000      1.00000
24 58 3.63689    osd.58      up 1.00000      1.00000
    59 3.63689    osd.59      up 1.00000      1.00000

```

Mit folgenden Befehlen kann man ein OSD vom Cluster entfernen. Dies sei am Beispiel des OSDs 59 gezeigt. Um alle OSDs des Ceph-Servers „osceph6“ zu entfernen, muss die Befehlsfolge für die anderen OSDs nur wiederholt werden.

Listing 11.3: Befehle zum Entfernen eines OSDs

```

# OSD 59 auf aus setzen
root@osceph2:~/# ceph osd out osd.59

# OSD-Daemon auf dem Ceph-Server osceph6 stoppen
5 root@osceph2:~/# ssh osceph6 "service ceph stop osd.59"

# OSD 59 aus der CRUSH-Map entfernen
root@osceph2:~/# ceph osd crush remove osd.59

10 # Authentifikationsschlüssel entfernen
root@osceph2:~/# ceph auth del osd.59

# OSD vom Cluster entfernen
root@osceph2:~/# ceph osd rm osd.59

```

Schließlich wird der ganze Ceph-Server „osceph6“ aus dem Cluster gelöst, nachdem alle seine OSDs entfernt worden sind.

Listing 11.4: Befehle zum Entfernen eines Ceph-Servers

```

1 # osceph6 vom Cluster entfernen
  root@osceph2:~/# ceph osd crush remove osceph6

  # Ceph auf osceph6 deinstallieren
  root@osceph2:~/# ceph-deploy purge osceph6
6
  # alle Ceph-Daten auf osceph6 löschen
  root@osceph2:~/# ceph-deploy purgedata osceph6

```

Man beachte, dass der Cluster-Status nun ausgibt, dass nur noch 48 OSDs zur Verfügung stehen. Auch die bereitstehende Speicherkapazität hat sich von ursprünglich 218 TB auf 174 TB verringert.

Listing 11.5: Clusterstatus nach Entfernung von osceph6

```

# gibt den Status und die Gesundheit eines Clusters an
2 root@osceph2:~/# ceph -s
  cluster 7bd91eb1-8196-41f0-87ca-e94cc3bf60f1
  health HEALTH_OK
  monmap e3: 3 mons at {osceph4=172.30.254.22:6789/0,osceph5=172.30.254.23:6789/0,
    osceph6=172.30.254.24:6789/0}
    election epoch 28, quorum 0,1,2 osceph4,osceph5,osceph6
7  osdmap e394: 48 osds: 48 up, 48 in
    flags sortbitwise,require_jewel_osds
  pgmap v12742: 0 pgs, 0 pools, 0 bytes data, 0 objects
    3021 MB used, 174 TB / 174 TB avail

```

11.2 Cluster vergrößern

Nun, da der Cluster nur noch aus vier Ceph-Servern besteht, soll gezeigt werden, dass es problemlos möglich ist, den Verbund um einen Ceph-Server zu erweitern, ihn also in die „Breite wachsen“ zu lassen und damit horizontal zu skalieren.

11.2.1 Horizontale Skalierbarkeit

Dazu wird der Befehl „ceph-deploy“ verwendet, der auch schon zum Erstellen des initialen Clusters zu Beginn dieser Arbeit benutzt worden ist. Folgende Kommandos installieren Ceph auf einem Rechner, formatieren die sechs für die OSDs ausersehenen Festplatten und gliedern sie samt Journalen auf der SSD „/dev/sdc“ in den Cluster ein.

Listing 11.6: Hinzufügen eines Cephserver mit 6 OSDs (mit SSD fürs Journal)

```

# Paketquellen von Ceph zu den Quellen von apt hinzufügen
root@osceph2:~/# ssh osceph6 "echo deb http://download.ceph.com/debian-jewel/ $(
    lsb_release -sc) main | sudo tee /etc/apt/sources.list.d/ceph.list"

# Paketquellen aktualisieren
5 root@osceph2:~/# ssh osceph6 "apt-get update"

# Ceph in der Version jewel auf osceph6 installieren
root@osceph2:~/# ceph-deploy install --release jewel osceph6

10 # für OSDs gewünschte Festplatten formatieren
root@osceph2:~/# ceph-deploy disk zap osceph6:sde osceph6:sdf osceph6:sdg osceph6:sdh
    osceph6:sdi osceph6:sdj

# OSDs und Journale (auf der ersten SSD /dev/sdc) vorbereiten (Partitionieren,
    Dateisystem aufspielen)
root@osceph2:~/# ceph-deploy osd prepare osceph6:sde:/dev/sdc osceph6:sdf:/dev/sdc
    osceph6:sdg:/dev/sdc osceph6:sdh:/dev/sdc osceph6:sdi:/dev/sdc osceph6:sdj:/dev/
    sdc

15 # OSDs und Journale aktivieren und in den Cluster einfügen
root@osceph2:~/# ceph-deploy osd activate osceph6:sde:/dev/sdc1 osceph6:sdf:/dev/sdc2
    osceph6:sdg:/dev/sdc3 osceph6:sdh:/dev/sdc4 osceph6:sdi:/dev/sdc5 osceph6:sdj:/dev
    /sdc6

```

Die Statusmeldung Ceph's zeigt unter „osdmap“ nach der Aktion an, dass sechs OSDs dem Cluster hinzugefügt worden und damit 54 OSDs verfügbar sind. Die Speicherkapazität beträgt nun 196 TB.

Listing 11.7: Clusterstatus nach Hinzufügen eines Cephserver und 6 OSDs

```

root@osceph2:~/# ceph -s
cluster 7bd91eb1-8196-41f0-87ca-e94cc3bf60f1
3 health HEALTH_OK
monmap e3: 3 mons at {osceph4=172.30.254.22:6789/0,osceph5=172.30.254.23:6789/0,
    osceph6=172.30.254.24:6789/0}
    election epoch 28, quorum 0,1,2 osceph4,osceph5,osceph6
osdmap e412: 54 osds: 54 up, 54 in
    flags sortbitwise,require_jewel_osds
8 pgmap v12845: 0 pgs, 0 pools, 0 bytes data, 0 objects
    3342 MB used, 196 TB / 196 TB avail

```

Die Baumansicht verrät, welcher Ceph-Server hinzugekommen ist („osceph6“) und welche neuen OSDs ihm angehören. Interessant ist zu sehen, dass das fiktive „Gesamtgewicht“ von „osceph6“ im Vergleich zu „osceph5“ bspw. nur 21.82132 beträgt statt 43.64264, was darauf zurückzuführen ist, dass dieser Ceph-Server ja zur Zeit nur aus sechs OSDs besteht, wohingegen „osceph5“ bereits über 12 OSDs verfügt.

Listing 11.8: Baumstruktur des Clusters nach Hinzufügen eines Cephserverns und 6 OSDs

```

1 # gibt die Baumstruktur eines Clusters an
  root@osceph2:~/# ceph osd tree
  ID WEIGHT  TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 218.21320 root default
-2 43.64264  host osceph2
6 ...
-3 43.64264  host osceph3
...
-4 43.64264  host osceph4
...
11 -5 43.64264  host osceph5
...
-6 21.82132  host osceph6
48 3.63689   osd.48      up 1.00000      1.00000
49 3.63689   osd.49      up 1.00000      1.00000
16 50 3.63689   osd.50      up 1.00000      1.00000
51 3.63689   osd.51      up 1.00000      1.00000
52 3.63689   osd.52      up 1.00000      1.00000
53 3.63689   osd.53      up 1.00000      1.00000

```

11.2.2 Vertikale Skalierbarkeit

So gut und einfach wie es möglich ist, einen Ceph-Server zum Cluster hinzuzufügen, so unproblematisch ist es auch, neue OSDs zu einem bestehenden Ceph-Server zu addieren. Diese Eigenschaft eines Storage-Systems nennt man auch „vertikale Skalierbarkeit“. Die Befehle, die dazu nötig sind, unterscheiden sich nicht von den vorigen. Wieder kommt „ceph-deploy“ zum Einsatz. Im Folgenden werden weitere sechs OSDs zum Ceph-Server „osceph6“ hinzugefügt und damit die vertikale Skalierbarkeit unter Beweis gestellt.

Listing 11.9: Hinzufügen von 6 OSDs (mit SSD fürs Journal) zu einem Ceph-Server

```

1 # für OSDs gewünschte Festplatten formatieren
  root@osceph2:~/# ceph-deploy disk zap osceph6:sd1 osceph6:sdm osceph6:sdn
    osceph6:sdo osceph6:sdp

# OSDs und Journale (auf der zweiten SSD /dev/sdd) vorbereiten (Partitionieren,
  Dateisystem aufspielen)
root@osceph2:~/# ceph-deploy osd prepare osceph6:sd1:/dev/sdd osceph6:sdm:/dev/sdd
  osceph6:sdn:/dev/sdd osceph6:sdo:/dev/sdd osceph6:sdp:/dev/
  sdd
6

# OSDs und Journale aktivieren und in den Cluster einfügen
root@osceph2:~/# ceph-deploy osd activate osceph6:sd1:/dev/sdd2
  osceph6:sdm:/dev/sdd3 osceph6:sdn:/dev/sdd4 osceph6:sdo:/dev/sdd5 osceph6:sdp:/dev
  /sdd6

```

Nach der Erweiterung um sechs OSDs zeigt der Clusterstatus an, dass dem Cluster nunmehr 60 OSDs zur Verfügung stehen und 218 TB Gesamtspeicher nutzbar sind.

Listing 11.10: Clusterstatus nach Hinzufügen von 6 OSDs

```
# gibt den Status und die Gesundheit eines Clusters an
2 root@osceph2:~/# ceph -s
  cluster 7bd91eb1-8196-41f0-87ca-e94cc3bf60f1
  health HEALTH_OK
  monmap e3: 3 mons at {osceph4=172.30.254.22:6789/0,osceph5=172.30.254.23:6789/0,
    osceph6=172.30.254.24:6789/0}
    election epoch 28, quorum 0,1,2 osceph4,osceph5,osceph6
7  osdmap e430: 60 osds: 60 up, 60 in
    flags sortbitwise,require_jewel_osds
  pgmap v12950: 0 pgs, 0 pools, 0 bytes data, 0 objects
    3695 MB used, 218 TB / 218 TB avail
```

Die Baumansicht zeigt, dass die sechs zusätzlichen OSDs dem Ceph-Server „osceph6“ zugeordnet worden sind und dessen Gewicht damit auf 43.64264 angewachsen ist.

Listing 11.11: Baumstruktur des Clusters nach vertikaler Skalierung

```
# gibt die Baumstruktur eines Clusters an
root@osceph2:~/# ceph osd tree
ID WEIGHT  TYPE NAME          UP/DOWN REWEIGHT PRIMARY-AFFINITY
-1 218.21320 root default
5  -2 43.64264  host osceph2
  ...
  -3 43.64264  host osceph3
  ...
  -4 43.64264  host osceph4
10 ...
  -5 43.64264  host osceph5
  ...
  -6 43.64264  host osceph6
48 3.63689   osd.48      up 1.00000      1.00000
15 49 3.63689   osd.49      up 1.00000      1.00000
50 3.63689   osd.50      up 1.00000      1.00000
51 3.63689   osd.51      up 1.00000      1.00000
52 3.63689   osd.52      up 1.00000      1.00000
53 3.63689   osd.53      up 1.00000      1.00000
20 54 3.63689   osd.54      up 1.00000      1.00000
55 3.63689   osd.55      up 1.00000      1.00000
56 3.63689   osd.56      up 1.00000      1.00000
57 3.63689   osd.57      up 1.00000      1.00000
58 3.63689   osd.58      up 1.00000      1.00000
25 59 3.63689   osd.59      up 1.00000      1.00000
```

11.3 Monitore und Metadata-Server

Auch die Anzahl von Monitoren und Metadata-Servern muss skalierbar sein, damit ein wachsender Cluster keinen Single-Point of Failure hat. Beide Server lassen sich einfach per

„ceph-deploy“ aufsetzen. Die Anzahl dieser Server ist im Cluster unbegrenzt. Wie weiter oben (siehe Abschnitt 2.1) schon beschrieben worden ist, ist dabei nur darauf zu achten, dass die Anzahl der Monitore stets ungerade ist, damit problemlos ein Quorum gebildet werden kann, um „Split-Brain“-Probleme zu vermeiden.

Folgende Befehle erzeugen Monitore und Metadata-Server (Achtung! Einschränkung siehe unten):

Listing 11.12: Erstellen zusätzlicher Monitore und Metadata-Server

```
# erstellt einen Monitor auf osceph2
root@osceph2:~/# ceph-deploy mon create osceph2

# erstellt einen Metadata-Server auf osceph2
5 root@osceph2:~/# ceph-deploy mds create osceph2
```

Bei der Erstellung weiterer Metadata-Server sei angemerkt, dass es zum Zeitpunkt der Erstellung dieser Arbeit in den Ceph-Docs noch nicht empfohlen wird, mehr als einen Metadata-Server zu betreiben [32]. Dieses Feature soll erst in späteren Versionen Ceph's verlässlich implementiert sein. Die Konsequenz daraus lautet daher schlicht, dass ein produktiver Betrieb des Ceph-Filesystems, welches den Metadataserver benötigt, zum aktuellen Zeitpunkt nicht empfohlen werden kann, da der Ausfall des einzigen Metadata-Servers die Nicht-Verfügbarkeit des gesamten Dateisystems zur Folge hätte!

11.4 Zusammenfassung

Die Verkleinerung eines Clusters ist nur eingeschränkt möglich, da die Anzahl der PGs eines Pools nicht mit Bordmitteln Ceph's verkleinert werden kann. Die Vergrößerung in vertikaler und horizontaler Richtung ist aber völlig unproblematisch und einfach zu bewerkstelligen.

Monitore können mit einem einzeiligen Befehl hinzugefügt werden. Die Erstellung von weiteren Metadata-Servern gelingt zwar mit dem vorgestellten Befehl, die Entwickler von Ceph raten aber derzeit noch von einem parallelen Betrieb mehrerer MDS ab, was die Nutzbarkeit des Ceph-Filesystems einschränkt.

Dennoch lassen sich mit den vorgestellten Methoden, bis auf die Metadata-Server, Cluster beliebig erweitern, den Entwicklern von Ceph nach, bis hinein in den Exabyte-Bereich [34].

Kapitel 12

Fazit und Ausblick

Die im Kapitel 2 angesprochenen Features, dass Ceph keinen „Single-Point-of-Failure“ habe, jede Komponente skalierbar und es auf heterogener Commodity-Hardware lauffähig sei sowie horizontal als auch vertikal skaliere, sind untersucht worden.

„Single-Point-of-Failure“ - Die Komponenten Ceph sind ausfallsicher. In Kapitel 10 wurde gezeigt, dass der Ausfall eines Ceph-Servers bzw. mehrerer OSDs die Weiterarbeit des Clusters nicht stark behindert. Bis auf eine ggfs. leicht sinkende Übertragungsgeschwindigkeit, sind Schreib- und Leseoperationen auch während eines Ausfalls und einer Recoveryprozedur möglich. Monitore und Metadata-Server können mehrfach installiert werden und stellen so keinen einzelnen Ausfallpunkt dar. Hier muss allerdings die Einschränkung gemacht werden, dass die Metadata-Server im derzeitigen Major-Release „jewel“ zwar schon redundant angelegt werden können, der Betrieb mehrerer Instanzen von den Ceph-Entwicklern jedoch noch nicht empfohlen wird [32].

Lauffähigkeit auf Commodity-Hardware - Ceph wurde auf einem Cluster von Rechnern relativ schwacher und schon etwas betagter Hardware installiert (Commodity-Hardware-Cluster) und mit den verwendeten Benchmark-Tools getestet. Wie im Kapitel 7.1 gezeigt wurde, sind dabei annehmbare Resultate erzielt worden. Damit ist gezeigt, dass Ceph auf dieser Art Hardware betrieben werden kann.

Ceph skaliert horizontal und vertikal - In Kapitel 11 wurde der Frage nachgegangen, ob ein bestehender Ceph-Cluster sowohl in die Breite (horizontal) als auch in die Höhe (vertikal) erweitert werden kann. Durch vergleichsweise einfache Befehle konnte dies am Beispiel des Hinzufügens eines Ceph-Servers (horizontale Skalierbarkeit) und mehrerer Festplatten bzw. OSDs zu einem bestehenden Ceph-Server (vertikale Skalierbarkeit) demonstriert werden. Auch für das einfache Hinzufügen von Monitoren und Metadata-Servern gibt es wirksame Kommandos und es wurde gezeigt, dass sie funktionieren. Für

den Betrieb mehrerer Metadata-Server existiert derzeit noch die erwähnte Beschränkung auf eine einzige laufende Instanz.

Ein weiteres Ziel dieser Arbeit war der Vergleich der beiden von Ceph angebotenen Speicherinterfaces „RADOS Block-Device“ und „Ceph-Filesystem“ hinsichtlich Geschwindigkeit und Lauffähigkeit. In den Kapiteln 7 und 8 wurden sie jeweils getestet und in den Grafiken des Kapitels 9.2 einander gegenübergestellt. Dabei hat sich gezeigt, dass das RADOS Block-Device leichte Geschwindigkeitsvorteile in der Konfiguration mit Filestore, Journal-SSDs und Replikationspools besitzt. In puncto Lesegeschwindigkeit übertrifft es das Ceph-Filesystem in dieser Konfiguration sogar deutlich. Bei der Nutzung des Bluestores als Backend liegen beide Techniken ca. gleich auf. Möchte man jedoch die Vorteile des Erasure-Codings benutzen, kann man beim derzeitigen Stand der Ceph-Entwicklung das Ceph-Filesystem nicht einfach verwenden (siehe Kapitel 8.3). Der Versuch, das Dateisystem auf Erasure-Coding Pools zu betreiben, schlug fehl. In den Ceph-Docs [30] steht zu lesen, dass dies erst in kommenden Versionen Ceph's verlässlich möglich sein wird. Darüber hinaus ist durch den derzeitig noch nicht garantierten Einsatz redundanter Instanzen von für das Dateisystem nötiger Metadata-Server [32], von einer produktiven Nutzung dieses Speicherinterfaces zum aktuellen Zeitpunkt abzuraten.

Das nächste Major-Release Ceph's wird „luminous“ sein. Der davon existierende Release-Candidate wurde in dieser Arbeit bereits eingesetzt, um das Erasure-Coding und das Bluestore-Backend zu testen. Es wird u.a. Verbesserungen beim Anlegen von Bluestores mit SSDs für RocksDB und WAL mit sich bringen. Des weiteren wird ein neuer Dienst „mgr“ (Ceph Manager Daemon) eingeführt, der bspw. ein natives Dashboard für das Monitoring des Ceph-Clusters über ein Web-Interface mit sich bringen und dann die zentrale Schnittstelle für alle externen Monitoring-Dienste sein wird. Zum jetzigen Zeitpunkt ist noch nicht absehbar, ob das Ceph-Filesystem in „luminous“ standardmäßig mit Erasure-Coding nutzbar sein wird und es dann möglich ist, mehrere Metadata-Server gleichzeitig einzusetzen.

Literaturverzeichnis

- [1] Karen Singh. *Learning Ceph*. Packt Publishing, 1 January 2015.
- [2] BlueStore - Präsentation von Sage Weil. <https://de.slideshare.net/sageweil/bluestore-a-new-storage-backend-for-ceph-one-year-in>, 25 August 2017.
- [3] CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. <https://ceph.com/wp-content/uploads/2016/08/weil-crush-sc06.pdf>, August 2016.
- [4] Fio man page. <https://linux.die.net/man/1/fio>, 17 August 2017.
- [5] Linux Magazin. <http://www.linux-mag.com/id/7744/>, 7 July 2017.
- [6] Wikipedia - Object Storage. https://en.wikipedia.org/wiki/Object_storage, 7 July 2017.
- [7] Karan Singh. *Ceph Cookbook*. Packt Publishing, 29 February 2016.
- [8] Ceph Online Dokumentation. <http://docs.ceph.com/docs/master/>, 7 July 2017.
- [9] Nick Fisk. *Mastering Ceph*. Packt Publishing, 30 May 2017.
- [10] Bluestore - Blog von Sebastian Han. <https://www.sebastien-han.fr/blog/2016/03/21/ceph-a-new-store-is-coming/>, 25 August 2017.
- [11] RocksDB für Bluestore. <https://www.golem.de/news/rocksdb-weil-googles-leveldb-fuer-facebook-zu-langsam-ist-1311-102990.html>, 25 August 2017.
- [12] CephFS-Possix. <http://docs.ceph.com/docs/jewel/cephfs/posix/>, 26 July 2017.
- [13] Thomas Krenn Wiki - Ceph. <https://www.thomas-krenn.com/de/wiki/Ceph>, 10 July 2017.
- [14] Placement Groups. https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/1.3/html/storage_strategies_guide/placement_groups_pgs, 27 July 2017.
- [15] PG Calculator. <http://ceph.com/pgcalc/>, 11 July 2017.

- [16] Ceph Docs Placement Groups. <http://docs.ceph.com/docs/jewel/rados/operations/placement-groups/>, 27 July 2017.
- [17] Ceph-Docs User Management. <http://docs.ceph.com/docs/jewel/rados/operations/user-management/>, 27 August 2017.
- [18] Collectd. <https://collectd.org/>, 13 July 2017.
- [19] Graphite. <https://graphiteapp.org/>, 12 July 2017.
- [20] Whisper Database. <http://graphite.readthedocs.io/en/latest/whisper.html>, 27 July 2017.
- [21] Bonnie++. <https://wiki.ubuntuusers.de/bonnie%2B%2B/>, 29 August 2017.
- [22] Bonding. https://en.wikipedia.org/wiki/Link_aggregation, 19 July 2017.
- [23] Ceph-intern Benchmarks. http://tracker.ceph.com/projects/ceph/wiki/Benchmark_Ceph_Cluster_Performance, 13 July 2017.
- [24] ZFS auf Linux/ fio. https://de.wikibooks.org/wiki/ZFS_auf_Linux/_fio, 17 August 2017.
- [25] Fio Enginge rbd. <https://github.com/axboe/fio/blob/master/examples/rbd.fio>, 17 August 2017.
- [26] Fio - terse output (minimal). https://www.andypeace.com/fio_minimal.html, 17 August 2017.
- [27] Ceph Docs - OSDs mit SSDs einrichten. <http://docs.ceph.com/docs/master/rados/deployment/ceph-deploy-osd/>, 25 August 2017.
- [28] Kernel 4.11 für Luminous und RBD nötig. <https://www.spinics.net/lists/ceph-users/msg37692.html>, 26 August 2017.
- [29] Ceph Docs - Bluestore. <http://docs.ceph.com/docs/master/rados/operations/erasure-code/#erasure-coding-with-overwrites>, 26 August 2017.
- [30] Ceph Dateisystem nicht fuer Erasure Coding. <https://www.spinics.net/lists/ceph-devel/msg23977.html>, 27 August 2017.
- [31] Übersicht über Features und Kernels. <http://cephnotes.ksperis.com/blog/2014/01/21/feature-set-mismatch-error-on-ceph-kernel-client>, 27 August 2017.
- [32] Ceph Docs - MDS keine Redundanz. <http://docs.ceph.com/docs/master/rados/deployment/ceph-deploy-mds/>, 27 August 2017.

- [33] PG-Anzahl kann nicht reduziert werden. <https://stackoverflow.com/questions/39589696/ceph-too-many-pgs-per-osd-all-you-need-to-know>, 28 August 2017.
- [34] Ceph - Exabyte Kapazität. <http://docs.ceph.com/docs/master/architecture/#architecture>, 28 August 2017.

Anhang A

Listings

Listing A.1: Script zur Installation und Konfiguration von collectd

```
#!/bin/bash

#####
## This Script was made from Joerg Broy for his Bachelor-Thesis ##
5 ## in 2017. ##
#####

#name of the ceph config file for collectd
conf_file_name="/etc/collectd/collectd.conf.d/collectd-ceph.conf"
10

#name of the processes config file for collectd
conf_file_name_processes="/etc/collectd/collectd.conf.d/collectd-processes.conf"

#test if root
15 if [ "$(id -u)" != "0" ]; then
    echo "this script must be run as root" 1>&2
    exit 1
fi

20 #Set Script Name variable
SCRIPT='basename ${BASH_SOURCE[0]}'

#Initialize variables to default values.
#OPT_D="all"
25

#Set fonts for Help.
NORM='tput sgr0'
BOLD='tput bold'
REV='tput smso'
30

#Help function
function HELP {
    echo -e "\n"Help documentation for ${BOLD}${SCRIPT}.${NORM}"\n
    echo -e "${REV}Basic usage:${NORM} ${BOLD}${SCRIPT} [Options]${NORM}"\n
35 echo "Command line switches are optional. The following switches are recognized."
    echo "${REV}-f${NORM} --shows the size of the whisper database."
    echo "${REV}-i${NORM} --makes apt-get update and installs collectd."
    echo "${REV}-a${NORM} --writes all plugin config files in /etc/collectd.collectd.conf
        .d/ and copies collectd.conf to /etc/collectd/."
    echo "${REV}-e${NORM} --copies collectd.conf to /etc/collectd/."
40 echo "${REV}-c${NORM} --writes ceph plugin config file to /etc/collectd.conf.d/."
    echo "${REV}-p${NORM} --writes processes plugin config file to /etc/collectd.conf.d
        /."
}
```

```

    echo "${REV}-s${NORM} --restart collectd daemon."
    echo "${REV}-r${NORM} --remove plugin from config file under /etc/collectd/collectd.
    conf. Values are: ceph, processes, all."
    echo "${REV}-l${NORM} --load plugin in config file under /etc/collectd/collectd.conf.
    Values are: ceph, processes, all."
45  echo "${REV}-x${NORM} --deletes the whisper database 'collectd' in graphite docker
    container."
    echo "${REV}-d${NORM} --purge collectd from node (with all data in all directories)."
    echo -e "${REV}-h${NORM} --Displays this help message. No further functions are
    performed."\\n
    echo -e "Example: ${BOLD}$SCRIPT -ai${NORM}"\\n
    exit 1
50 }

#write ceph config file
function CEPH-CONFIG {
    #get number of osds
55  numberOfOSDs=$(ls -la /var/run/ceph | grep osd | wc -l)
    numberOfOSDsZero=$((numberOfOSDs-1))
    echo $numberOfOSDsZero " OSDs found on this machine."

    #fill arrays with osd_names, osd_sockets and osd_nums
60  unset osd_names
    unset osd_sockets
    unset osd_nums
    for f in /var/run/ceph/*; do
        one_osd_socket=$(echo $f | grep osd)
        osd_sockets+=($one_osd_socket)
65  one_osd=$(echo $f | grep osd | cut -d- -f2 | cut -d. -f1,2)
        osd_names+=($one_osd)
        one_osd_num=$(echo $f | grep osd | cut -d- -f2 | cut -d. -f2)
        osd_nums+=($one_osd_num)
70  done

    #fill arrays with mon_names and mon_sockets
    unset mon_names
    unset mon_sockets
75  for f in /var/run/ceph/*; do
        one_mon_socket=$(echo $f | grep mon)
        mon_sockets+=($one_mon_socket)
        one_mon=$(echo $f | grep mon | cut -d- -f2 | cut -d. -f1,2)
        mon_names+=($one_mon)
80  done

    #fill arrays with mds_names and mds_sockets
    unset mds_names
    unset mds_sockets
85  for f in /var/run/ceph/*; do
        one_mds_socket=$(echo $f | grep mds)
        mds_sockets+=($one_mds_socket)
        one_mds=$(echo $f | grep mds | cut -d- -f2 | cut -d. -f1,2)
        mds_names+=($one_mds)
90  done

    #####
    ## create and write config file for ceph plugin ##
    #####
95  echo "" > $conf_file_name
    echo "<Plugin ceph>" >> $conf_file_name
    echo "  LongRunAvgLatency false" >> $conf_file_name
    echo "  ConvertSpecialMetricTypes true" >> $conf_file_name

100 #write all osd daemons in the config file

```

```

    for index in ${!osd_names[*]}; do
        echo "    <Daemon "'${osd_names[$index]}'">" >> $conf_file_name
        echo "        SocketPath "'${osd_sockets[$index]}'" >> $conf_file_name
        echo "    </Daemon>" >> $conf_file_name
105 done

    #write all mon daemons in the config file
    for index in ${!mon_names[*]}; do
        echo "    <Daemon "'${mon_names[$index]}'">" >> $conf_file_name
        echo "        SocketPath "'${mon_sockets[$index]}'" >> $conf_file_name
110     echo "    </Daemon>" >> $conf_file_name
    done

    #write all mds daemons in the config file
115 for index in ${!mds_names[*]}; do
        echo "    <Daemon "'${mds_names[$index]}'">" >> $conf_file_name
        echo "        SocketPath "'${mds_sockets[$index]}'" >> $conf_file_name
        echo "    </Daemon>" >> $conf_file_name
    done
120 echo "</Plugin>" >> $conf_file_name
}

#write processes config file
function PROCESSES_CONFIG {
125
    #fill arrays with osd_names, osd_sockets and osd_nums
    unset osd_names
    unset osd_sockets
    unset osd_nums
130 for f in /var/run/ceph/*; do
        one_osd_socket=$(echo $f | grep osd)
        osd_sockets+=($one_osd_socket)
        one_osd=$(echo $f | grep osd | cut -d- -f2 | cut -d. -f1,2)
        osd_names+=($one_osd)
135     one_osd_num=$(echo $f | grep osd | cut -d- -f2 | cut -d. -f2)
        osd_nums+=($one_osd_num)
    done

    #####
140 ## create and write processes config file ##
    #####

    echo "" > $conf_file_name_processes
    echo "<Plugin processes>" >> $conf_file_name_processes
145 echo "    Process "'ceph-osd'" >> $conf_file_name_processes
    echo "    Process "'ceph-mon'" >> $conf_file_name_processes

    for index in ${!osd_names[*]}; do
        echo "        ProcessMatch \"${osd_names[$index]}\" \"\`/usr/bin/ceph-osd.*${osd_nums[$index]}.*\"" >> $conf_file_name_processes
150     done
    echo "</Plugin>" >> $conf_file_name_processes
}

#test if file "collectd.conf" is in the right directory:
155 function TEST_FILE_IN_DIR {
    if [ -e "/root/my-cluster/scripts/collectd.conf" ]
    then
        echo "All right. Found collectd.conf in /root/my-cluster/scripts/"
    else
160     echo "file collectd.conf needs to be in /root/my-cluster/scripts/"
        exit 1
    fi
}

```

```

}

165 #Check the number of arguments. If none are passed, print help and exit.
    NUMARGS=$#
    #echo -e \\n"Number of arguments: $NUMARGS"
    if [ $NUMARGS -eq 0 ]; then
        HELP
170 fi

while getopts :fiaecpsr:l:x:d:h FLAG; do
    case $FLAG in
        f) #set option "f"; show size of whisper database
175         ssh root@sellerie du -hs /NOBACKUP/graphite/whisper/collectd
            ;;
        i) #set option "i"; install collectd
            apt-get update
            apt-get -y install collectd
180         echo "Installation succesfull"
            ;;
        a) #set option "a"; write all config files and overwrite collectd.conf
            TEST_FILE_IN_DIR
            CEPH_CONFIG
185         PROCESSES_CONFIG
            #copy collectd.conf from working directory into collectd directory
            mv /etc/collectd/collectd.conf /etc/collectd/collectd.conf.old
            cp /root/my-cluster/scripts/collectd.conf /etc/collectd/collectd.conf
            echo "All config files written and collectd.conf copied to /etc/collectd/"
190         ;;
        e) #set option "e"; copies collectd.conf to /etc/collectd/
            TEST_FILE_IN_DIR
            #copy collectd.conf from working directory into collectd directory
            mv /etc/collectd/collectd.conf /etc/collectd/collectd.conf.old
195         cp /root/my-cluster/scripts/collectd.conf /etc/collectd/collectd.conf
            echo "collectd.conf copied to /etc/collectd/"
            ;;
        c) #set option "c"; write ceph plugin config file
            CEPH_CONFIG
200         echo "Ceph config file written to /etc/collectd/collectd.conf.d/"
            ;;
        p) #set option "p"; write processes plugin config file
            PROCESSES_CONFIG
            echo "Processes config file written to /etc/collectd/collectd.conf.d/"
205         ;;
        s) #set option "s"; restart collectd
            service collectd stop
            service collectd start
            echo "Daemon succesfully restarted."
210         ;;
        r) #set option "r"; remove certain plugins
            case "$OPTARG" in
                ceph) sed -i 's/.*LoadPlugin ceph/#LoadPlugin ceph/' /etc/collectd/collectd.
                    conf; cat /etc/collectd/collectd.conf
                    ;;
215                 processes) sed -i 's/.*LoadPlugin processes/#LoadPlugin processes/' /etc/
                    collectd/collectd.conf; cat /etc/collectd/collectd.conf
                    ;;
                all) sed -i 's/.*LoadPlugin ceph/#LoadPlugin ceph/' /etc/collectd/collectd.conf
                    ; sed -i 's/.*LoadPlugin processes/#LoadPlugin processes/' /etc/collectd/
                    collectd.conf; cat /etc/collectd/collectd.conf
                    ;;
                *) echo "Wrong Parameter given. Use \"-d [{ceph}{processes}{all}]\""
220             esac
            ;;
    esac

```

```

1) #set option "l"; load certain plugins
    case "$OPTARG" in
        ceph) sed -i 's/.*LoadPlugin ceph/LoadPlugin ceph/' /etc/collectd/
                collectd.conf; cat /etc/collectd/collectd.conf
225         ;;
        processes) sed -i 's/.*LoadPlugin processes/LoadPlugin processes/' /
                etc/collectd/collectd.conf; cat /etc/collectd/collectd.conf
                ;;
        all) sed -i 's/.*LoadPlugin ceph/LoadPlugin ceph/' /etc/collectd/
                collectd.conf; sed -i 's/.*LoadPlugin processes/LoadPlugin
                processes/' /etc/collectd/collectd.conf; cat /etc/collectd/
                collectd.conf
                ;;
230     *) echo "Wrong Parameter given. Use \"-e [{ceph}{processes}{all}]\""
        esac
        ;;
x) #set option "x"; deletes the whisper database of collectd in graphite docker
    container
    if [ "$OPTARG" == "yes-i-do-will" ]
235    then
        docker exec -i -t graphite rm -rf /opt/graphite/storage/whisper/
                collectd
        echo "Deleting of whisper database in graphite docker container
                successfull."
    else
        echo "Not deleted anything. Usage: go-collectd.sh -x yes-i-do-will"
240    fi
        ;;
d) #set option "d"; removes collectd and all its data from the node
    if [ "$OPTARG" == "yes-i-do-will" ]
    then
245        apt-get purge -y collectd
        rm -rf /etc/collectd
        echo "Complete deleting of collectd successfull."
    else
        echo "Not deleted anything. Usage: go-collectd.sh -d yes-i-do-will"
250    fi
        ;;
h) #show help
    HELP
    ;;
255 \?) #unrecognized option - show help
    echo -e "\nOption -${BOLD}$OPTARG${NORM} not allowed."
    HELP
    #If you just want to display a simple error message instead of the full
    #help, remove the 2 lines above and uncomment the 2 lines below.
260    #echo -e "Use ${BOLD}$SCRIPT -h${NORM} to see the help documentation."\n
    #exit 2
    ;;
:) echo "Missing option argument for -$OPTARG" >&2; exit 1
    ;;
265 esac
done

```

Listing A.2: Ausgangskonfigurationsdatei von collectd

```

#####
2 ## This Config was made from Joerg Broy for his Bachelor-Thesis ##
## in 2017. ##
#####

#####
7 # Global #

```

```

#-----#
# Global settings for the daemon.                                     #
#####

12 FQDNLookup true

# intervall of measurement in seconds
Interval 1

17 LoadPlugin syslog
#LoadPlugin ceph
LoadPlugin cpu
LoadPlugin interface
LoadPlugin memory
22 #LoadPlugin processes
LoadPlugin write_graphite

#####
# Plugin configuration                                             #
#-----#
27 # In this section configuration stubs for each plugin are provided. A desc- #
# ription of those options is available in the collectd.conf(5) manual page. #
#####

32 <Plugin syslog>
    LogLevel info
</Plugin>

<Plugin interface>
37     #Exchange this Interface with yours.
    Interface "bond0"
    IgnoreSelected false
</Plugin>

42 <Plugin write_graphite>
    <Node "graphing">
        #Exchange this Hostname with yours.
        Host "oscomp5"
        Port "2003"
47         Protocol "tcp"
        LogSendErrors true
        Prefix "collectd."
        # Postfix "collectd"
        StoreRates true
52         AlwaysAppendDS false
        EscapeCharacter "_"
    </Node>
</Plugin>

57 <Include "/etc/collectd/collectd.conf.d">
    Filter "*.conf"
</Include>

```

Listing A.3: CRUSH placement for Object x [3]

```

procedure TAKE( $a$ )           ▷ Put item  $a$  in working vector  $\vec{i}$ 
   $\vec{i} \leftarrow [a]$ 
end procedure

procedure SELECT( $n, t$ )      ▷ Select  $n$  items of type  $t$ 
   $\vec{o} \leftarrow \emptyset$         ▷ Our output, initially empty
  for  $i \in \vec{i}$  do             ▷ Loop over input  $\vec{i}$ 
     $f \leftarrow 0$              ▷ No failures yet
    for  $r \leftarrow 1, n$  do    ▷ Loop over  $n$  replicas
       $f_r \leftarrow 0$         ▷ No failures on this replica
       $retry\_descent \leftarrow false$ 
      repeat
         $b \leftarrow bucket(i)$   ▷ Start descent at bucket  $i$ 
         $retry\_bucket \leftarrow false$ 
        repeat
          if “first  $n$ ” then    ▷ See Section 3.2.2
             $r' \leftarrow r + f$ 
          else
             $r' \leftarrow r + f_r n$ 
          end if
           $o \leftarrow b.c(r', x)$   ▷ See Section 3.4
          if  $type(o) \neq t$  then
             $b \leftarrow bucket(o)$   ▷ Continue descent
             $retry\_bucket \leftarrow true$ 
          else if  $o \in \vec{o}$  or  $failed(o)$  or  $overload(o, x)$ 
            then
               $f_r \leftarrow f_r + 1, f \leftarrow f + 1$ 
              if  $o \in \vec{o}$  and  $f_r < 3$  then
                 $retry\_bucket \leftarrow true$           ▷ Retry
                collisions locally (see Section 3.2.1)
              else
                 $retry\_descent \leftarrow true$       ▷ Otherwise
                retry descent from  $i$ 
              end if
            end if
          until  $\neg retry\_bucket$ 
          until  $\neg retry\_descent$ 
           $\vec{o} \leftarrow [\vec{o}, o]$           ▷ Add  $o$  to output  $\vec{o}$ 
        end for
      end for
    end for
     $\vec{i} \leftarrow \vec{o}$           ▷ Copy output back into  $\vec{i}$ 
  end procedure

procedure EMIT              ▷ Append working vector  $\vec{i}$  to result
   $\vec{R} \leftarrow [\vec{R}, \vec{i}]$ 
end procedure

```

Anhang B

CC-Lizenz

Die Arbeit wurde unter der Lizenz „Creative Commons Namensnennung-Weitergabe unter gleichen Bedingungen Deutschland“ in Version 3.0 (abgekürzt „CC-by-sa 3.0/de“) veröffentlicht.

Den rechtsverbindlichen Lizenzvertrag finden Sie unter

[http : //creativecommons.org/licenses/by – sa/3.0/de/legalcode](http://creativecommons.org/licenses/by-sa/3.0/de/legalcode).

Es folgt eine vereinfachte Zusammenfassung des Vertrags in allgemeinverständlicher Sprache ohne juristische Wirkung.

Es ist Ihnen gestattet,

- das Werk zu vervielfältigen, zu verbreiten und öffentlich zugänglich zu machen (**Weiterverwendung erlaubt**) sowie
- Abwandlungen und Bearbeitungen des Werkes anzufertigen (**Bearbeitung erlaubt**),

sofern Sie folgende Bedingungen einhalten:

- **Namensnennung:** Sie müssen den Urheber bzw. den Rechteinhaber in der von ihm festgelegten Weise, die URI (z. B. die Internetadresse dieser Seite) sowie den Titel des Werkes und bei einer Abwandlung einen Hinweis darauf angeben.
- **Weitergabe unter gleichen Bedingungen:** Wenn Sie das lizenzierte Werk bearbeiten, abwandeln oder als Vorlage für ein neues Werk verwenden, dürfen Sie die neu entstandenen Werke nur unter dieser oder einer zu dieser kompatiblen Lizenz nutzen und weiterverbreiten.
- **Lizenzangabe:** Sie müssen anderen alle Lizenzbedingungen mitteilen, die für dieses Werk gelten. Am einfachsten ist es, wenn Sie dazu einen Link auf den Lizenzvertrag (siehe oben) einbinden.

Bitte beachten Sie, dass andere Rechte die Weiterverwendung einschränken können.

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht, noch einer anderen Prüfungsbehörde vorgelegt.
