Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

Object Storage

vorgelegt von

Alexander Timmermann

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen
Proseminar „Speicher- und Dateisysteme”

Matrikelnummer: 6524072

Inhaltsverzeichnis

1 Einleitung

1.1 Begriffserklärungen

1.1.1 Blob

Ein Blob ist eine Kollektion von unstrukturierten Binärdaten. Der Begriff ist heute ein „backronym“ für *Binary Large Object*. Meist steht er allgemein für einen unstrukturierten Datensatz von relevanter Größe.

1.1.2 POSIX

POSIX steht für *Portable Operating System Interface* und ist eine Kollektion von Standards der *IEEE Computer Society*, die Kompatibilität zwischen Betriebssystemen herstellen sollen. [IEE18]

POSIX definiert vor allem Semantiken, die in einem HPC-Kontext ungünstige Auswirkungen auf die Performance haben können, wie z.B. dass alle Änderungen für alle parallelen Prozesse sofort sichtbar sein müssen.

Im Vergleich zu einer bei Object-Storage-Systemen üblichen HTTP API bietet die POSIX-Semantik für Dateioperationen deutlich mehr Funktionen, jedoch auch deutlich mehr Overhead.

1.2 Klassifikationen von Speichersystemen

Grob lassen sich moderne Speichersysteme in drei Kategorien einteilen:

- Block-basiert,
- Datei-basiert,
- Objekt-basiert.

Bei blockbasierten Speichersystemen findet ein gepufferter Zugriff auf Systemressourcen statt. Lese- und Schreibvorgänge können mit Blöcken beliebiger Größe durchgeführt werden, und ein dafür zuständiges System (meistens der Kernel des Betriebssystems) kümmert sich um den spezifischen Zugriff auf die eigentliche Hardware. Dadurch wird eine (im Vergleich relativ kleine) Abstraktionsebene geschaffen, die kleinste Speichereinheit ist ein Block.

Dateibasierte Systeme bauen meist auf Blockspeichern auf und organisieren die Blöcke anhand von Metadaten in eine Datei- und Ordnerstruktur. Zusätzlich werden bei den meisten Dateisystemen weitere Metadaten erfasst, wie z.B. der letzte Zugriff, Zugriffsberechtigungen, der Zeitpunkt der letzten Modifikation etc. Aufbauend auf den Blockspeichern wird hier also eine weitere Abstraktionsebene geschaffen.

2 Object Storage

Eine im, Vergleich zu den anderen beiden Klassifikationen, recht neue Art der Speichersysteme sind die objektbasierten Systeme (*object storage*). Als kleinste Speichereinheit dient hier das Objekt. Objekte sind *blobs* (s.) mit variablem Inhalt, einem fixen, vom Inhalt abhängigen Identifier und separat gespeicherten Metadaten, die in einem flachen Namespace abgelegt werden, d.h. ohne Ordnerstruktur. Bei manchen Implementationen (bspw. RADOS, S3) können Objekte zusätzlich aber in sog. *pools* bzw. *buckets* organisiert werden, die jedoch die einzige Art der hierarchischen Struktur darstellen.

Ein Object Storage System kann auf verschiedenen Ebenen implementiert werden:

1. der Geräte-Ebene,
2. der System-Ebene,
3. und der Interface-Ebene.

Zur Geräte-Ebene zählen vor allem fertig konfigurierte Object Storage Appliances, die vom Hersteller mit passender Firmware und/oder Software ausgeliefert werden, und welche ohne großen userseitigen Aufwand schnell eingesetzt werden können. Große Hersteller auf diesem Gebiet sind unter Anderem *DDN* und *Hitachi Data Systems*.

Der größte Vorteil, den Objektspeichersysteme momentan auf dem Markt bieten, ist, dass sie große Mengen von unstrukturierten Datensätzen performant speichern und vorhalten können. So hat Facebook z.B. sein eigenes Object Storage System entwickelt (Projektname *Haystack*), um den Anforderungen um die großen Mengen von Benutzerfotos gerecht zu werden. [Vaj09] Auch andere moderne Internetfirmen benutzen Objektspeicher um große Datenmengen zu verwalten, z.B. Spotify für Songs und Dropbox für alle Dateien der Nutzer.

Oft werden Objektspeichersysteme mit den sogenannten „Key-Value-Stores“ verglichen. Während die Systeme sich zwar ähneln, gibt es ein paar wichtige Unterschiede:

- Bei Object Storage sind die Metadaten ein wichtiger Teil des Objekts, während KV-Stores definitionsgemäß nur eine Einheit aus Key und Value zulassen.

- Object Stores sind häufig optimiert für große Datenmengen, während KV-Stores oft eine Begrenzung des Werts im Kilobyte-Bereich haben.

- In Object Storage Systems sind die Konsistenzkriterien häufig geringer als in KV-Stores.

  In KV-Stores wird fast immer eine starke Konsistenz umgesetzt, d.h. dass alle Zugriffe von allen parallelen Prozessen in der gleichen Reihenfolge gesehen werden und somit stets ein konsistenter Zustand entsteht.

  In Object Storage implementiert man aus Performance-Gründen jedoch häufig nur *eventual consistency*, d.h. ein Datensatz ist nach einer hinreichenden Zeit ohne Zugriffe oder Fehler „irgendwann“ konsistent.

2.1 Vor- und Nachteile

Die Vorteile von Object Storage sind vielfältiger Natur:

- **Hoher Durchsatz:** Durch die dezentralisierte Arbeitsweise der meisten Object Storage Systeme kann eine hohe Performance beim Schreiben und Lesen erreicht werden.

- **Hohe Skalierbarkeit der Kapazität:** Nahezu alle Object Storage Systeme erlauben es, die Kapazität einfach zu skalieren, indem neue Knoten zu einem Cluster hinzugefügt werden.

- **Zugriffs- und Berechtigungseinstellungen auf Objekt-Ebene:** Für jedes Objekt kann eingestellt werden, welche Zugriffe und Berechtigungen für verschiedene Rollen erlaubt sein sollen.

- **Metdataten separat und anpassbar:** Die Metadaten der Objekte werden meist nicht auf den selben Knoten wie die eigentlichen Objekte gespeichert. Damit kann eine Entkoppelung stattfinden, die wiederum Auswirkungen auf die Performance hat. Des weiteren sind die Metadaten anpassbar, sodass eigene, applikationsspezifische Definitionen von Key-Value-Paaren in die Metadaten mit eingefügt werden können.

Ein Nachteil dieser hohen Abstraktion und Performance ist hingegen auch die hohe Komplexität der am Ende umgesetzten Systeme.

2.2 Beispiele

2.2.1 Ceph/RADOS

Abbildung 2.1: Ceph Logo

Ceph ist eine Software-Plattform für verteilten Speicher, die auf RADOS, dem *Reliable Autonomic Distributed Object Storage*, aufbaut. Es wird als freie Software entwickelt und stellt Interfaces für alle oben beschriebenen Speicherklassifikationen bereit. Ziel des Projekts ist es, eine frei verfügbare Storageplattform bereitzustellen, die keinen *single point of failure* hat und bis zum Exabyte-Level skalieren kann.

Ceph zielt darauf ab, Administrationsaufwand zu minimieren: Die Architektur ist in den meisten operativen Szenarios sowohl selbst-heilend als auch selbst-verwaltend, d.h. die meisten administrativen Optimierungs- und Reperaturaufgaben können vom System selbst übernommen werden.

Wie Abbildung 2.2 zu entnehmen ist, baut Ceph für jeglichen Speicher auf RADOS auf. Dort kann entweder mit Bibliotheken direkt auf die API zugegriffen werden, oder es wird einer der Layer, die Ceph bereitstellt, genutzt. Diese Layer sind:

RADOSGW: RADOSGW ist eine REST-API, die die Funktionen von RADOS kompatibel zur API von Amazon S3 macht und damit Kompatibilität zu vielen S3-kompatiblen Bibliotheken herstellt.

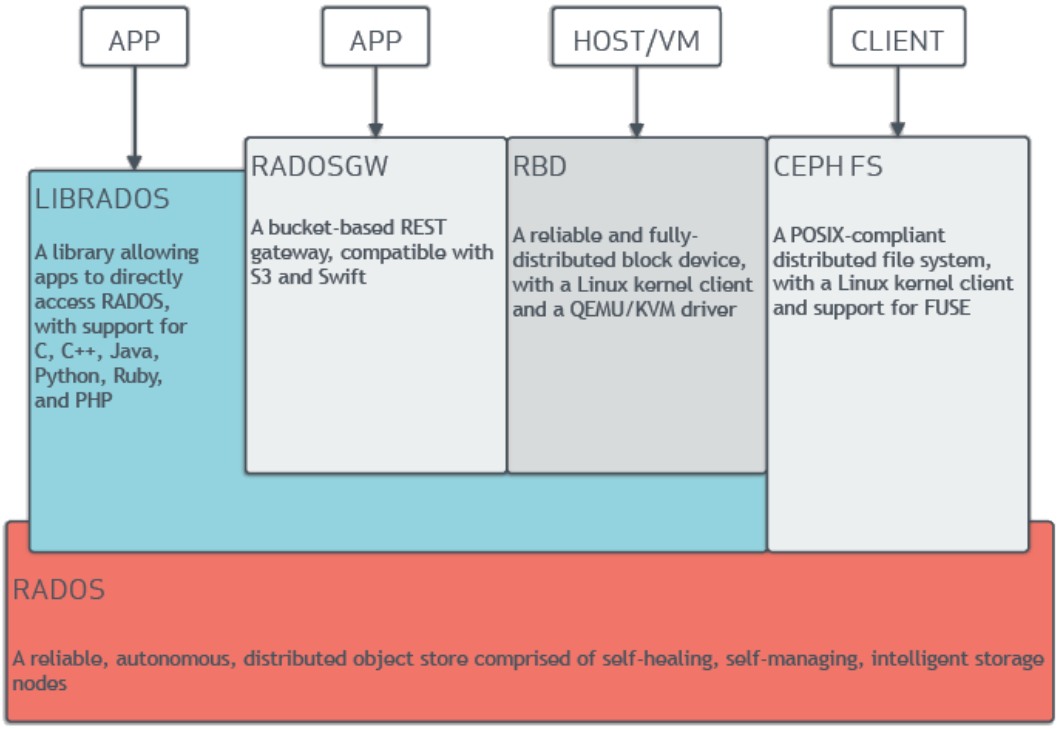

Abbildung 2.2: Eine Übersicht des Stacks, den Ceph bereitstellt. [McG15]

RBD: RBD stellt ein verteiltes Block Storage Device bereit. Es besitzt einen Client im Linux-Kernel und kann in der Virtualisierung für KVM bereitgestellt werden.

CephFS: CephFS stellt ein POSIX-kompatibles, verteiltes Dateisystem bereit. Ein Linux-Kernel-Treiber und Support für FUSE sind ebenfalls vorhanden.

Ein Ceph-Cluster besteht aus mehreren Komponenten, die jeweils mindestens einmal vorkommen müssen [Jon10] und in Abb. 2.3 schematisch dargestellt sind:

- *Client*:
  Der Client ist, je nach Zugriffsmethode, unterschiedlich gelayered. Am Ende steht jedoch immer eine Ceph-Instanz die Zugriffe auf Objekte an den *cosd* und Zugriffe auf Metadaten an den *cmds* weiterleitet.

- *cosd* (Ceph Object Storage daemon):
  Der *cosd* ist für die eigentliche Speicherung der Objekte in RADOS bzw. auf den Speichermedien zuständig. Er wird von allen anderen Komponenten angesprochen: vom Client, um Objekte zu lesen und zu schreiben; vom *cmds* um Metadaten zu

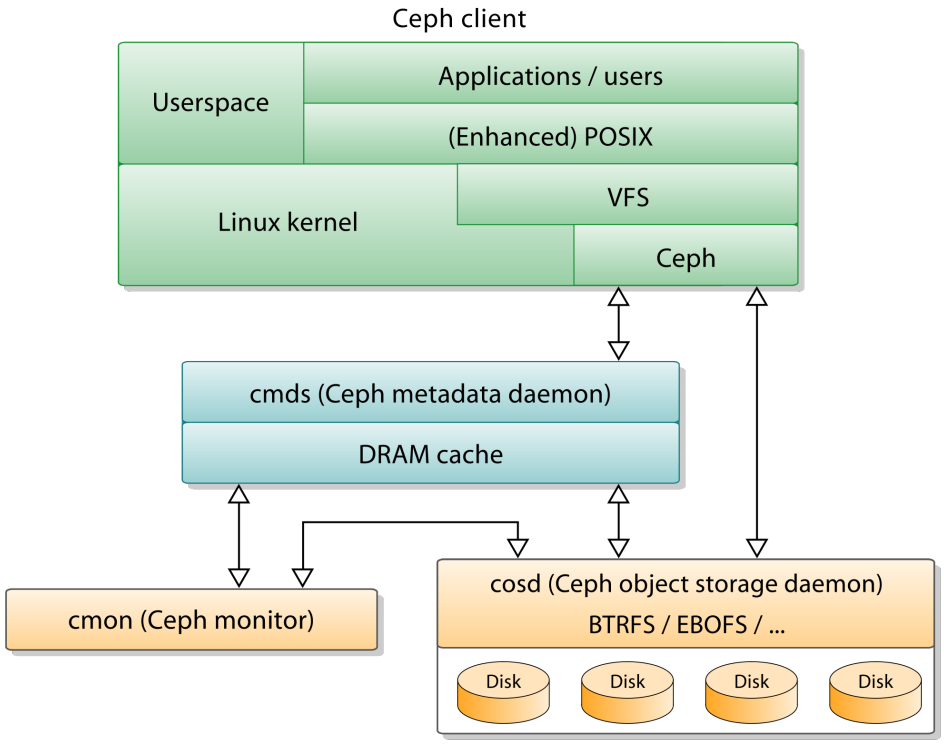

Abbildung 2.3: Die einzelnen Komponenten eines Ceph-Clusters.

verknüpfen; und vom *cmon* um den Status zu überwachen.

- *cmds* (Ceph Metadata Storage daemon):
  Der *cmds* ist für die Speicherung der Metadaten und die Verknüpfung dieser Metadaten zu den Objekten zuständig. Dafür besitzt er Cache, in dem die Metadaten zunächst zwischengespeichert werden, um Performance zu gewährleisten. den *cosd* übergeben werden.

- *cmon* (Ceph Monitor):
  Der *cmon* kümmert sich um die Stabilität und Optimierung des Clusters. Insbesondere überwacht er, welche Knoten aktiv sind und welche defekt sind und aus dem Cluster entfernt werden müssen.

2.2.2 Lustre

Lustre ist ein verteiltes, paralleles Dateisystem, das vor allem in großen Clustern Verwendung findet. Als unterliegende Architektur wird ebenfalls Object Storage verwendet, die jedoch für den Client als POSIX-kompatibles Dateisystem präsentiert wird.

Abbildung 2.4: Lustre Logo

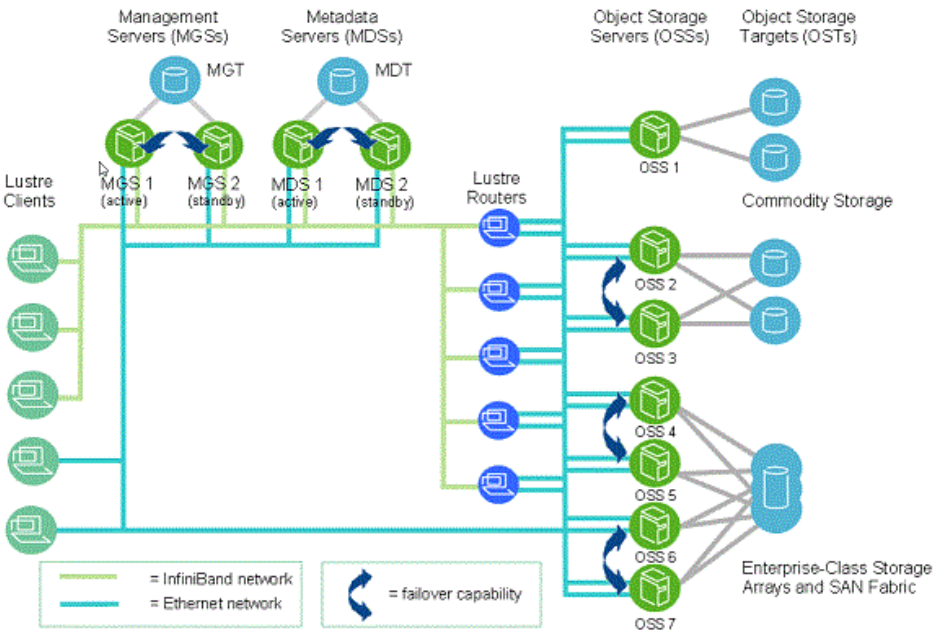

Abbildung 2.5: Die einzelnen Komponenten eines Lustre-Clusters.

Abb. 2.5 zeigt den Aufbau eines exemplarischen Lustre-Clusters. Interessant ist auch hier der zu Ceph ähnliche Aufbau aus Management-Servern (vergleichbar zu *cmon*), Metadaten-Servern (wie *cmds*) und Storage-Servern und -Targets (ähnlich wie *cosd*). Zusätzlich werden hier jedoch auch Routing-Server eingesetzt, die die gleichmäßige Auslastung der Storage-Knoten steuern.

2.2.3 BlobSeer

BlobSeer ist ein Forschungsprojekt von *Inria*, das auf Blobs basiert und Größen im Terabyte-Bereich verarbeiten kann. Es erzeugt atomare Snapshots, sodass nach jeder

Operation immer ein Rollback stattfinden kann. Der Schreibzugriff ist sehr granular möglich (bis in den Megabyte-Bereich genau) und das System kann als Storage-Backend in Hadoop eingebunden werden.

Die Architektur bietet jedoch einen *single point of failure*: den Versions-Manager, ohne welchen eine Operationen durchgeführt werden können. Des weiteren werden alte Snapshots nicht automatisch verworfen, nach längerer Benutzung kommt es also zu einem Performance-Verlust.

2.2.4 Týr

Týr ist ebenfalls ein Forschungsprojekt, das Blob Storage mit Transaktionen kombiniert und dadurch höhere Konsistenz gewährleisten kann. Das Datenmodell ist ähnlich zu RADOS jedoch ergänzt um Transaktions-Semantik und in-place Updates.[Mat+16]

Literatur

[Fac+] M. Factor u. a. “Object Storage: The Future Building Block for Storage Systems A Position Paper”. In: *2005 IEEE International Symposium on Mass Storage Systems and Technology*. IEEE. DOI: 10.1109/lgdi.2005.1612479.

[IEE18] IEEE. “IEEE Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7”. In: *IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)* (Jan. 2018), S. 1–3951. DOI: 10.1109/IEEESTD.2018.8277153.

[Jon10] M. Tim Jones. “Ceph: A Linux petabyte-scale distributed file system. Exploring the Ceph file system and ecosystem”. In: *IBM developerWorks* (2010). URL: https://www.ibm.com/developerworks/library/l-ceph/l-ceph-pdf.pdf (besucht am 20. 02. 2018).

[Mat+16] Pierre Matri u. a. “Týr: Blob Storage Meets Built-In Transactions”. In: *IEEE ACM SC16 - The International Conference for High Performance Computing, Networking, Storage and Analysis 2016*. Salt Lake City, United States, Nov. 2016. URL: https://hal.inria.fr/hal-01347652.

[Mat+17] Pierre Matri u. a. “Could Blobs Fuel Storage-Based Convergence Between HPC and Big Data?” In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, Sep. 2017. DOI: 10.1109/cluster.2017.63.

[McG15] Patrick McGarry. *Ceph Stack*. 2015. URL: https://github.com/ceph/ceph/blob/master/doc/images/stack.png (besucht am 25. 02. 2018).

[Nic+11] Bogdan Nicolae u. a. “BlobSeer: Next-generation data management for large scale infrastructures”. In: *Journal of Parallel and Distributed Computing* 71.2 (Feb. 2011), S. 169–184. DOI: 10.1016/j.jpdc.2010.08.004.

[Vaj09] Peter Vajgel. “Needle in a haystack: efficient storage of billions of photos”. In: *Facebook Code* (20. Apr. 2009). URL: https://code.facebook.com/posts/685565858139515/needle-in-a-haystack-efficient-storage-of-billions-of-photos/ (besucht am 20. 02. 2018).