Verteilte Filesysteme

Ceph, GlusterFS, Client-Sync (DiscoFS), ...

Jochen Schulz, Ralph Krimmel, Max Voit, Piotr Kasprzak

Georg-August Universität Göttingen

Inhalt

Aufbau

1. **Einführung und Motivation**

2. **Parallel, distributed, fault-tolerant filesystems**
   - Auswahl verteilte Dateisysteme
   - Ceph Theorie
   - Ceph Praxis

3. **Erfahrungen**

Philosophie und gewünschtes Setup

A: Synchron

- Compute-server: (Fokus: Compute)
- Clients: Laptops mit remote gemounteten Home (Fokus: tägliche Arbeit)
- (File-)Server: (Fokus: shared ressources)

B: Asynchron

- Daten stets (auch) lokal.
- Asynchroner Datenabgleich mit remote storage (dropbox,powerfolder,git ...)

Philosophie und gewünschtes Setup

Prinzipielle Vor- und Nachteile

- A: Flexibilität im Ort und Gerät (Keine nennenswerte Transferzeit)
- A: Parallelität
- A: Homogenität und Synchrones Arbeiten (z.B. lokales Editieren , remote Ausführen)
- B: Mobilität des Geräts
- B: Daten stets lokal (aber grössere Transferzeit)

Details bestehendes Setup (Typ A)

- clients ($\approx$ 50): Linux-Laptops, Datensync per unison
- Login Server (2): remote Desktop, ssh, Datentransfer
- Compute Server (6): remote Desktop, ssh, Datentransfer, FHGFS, WiRe, CUDA
- Fileserver (2): (kein SAN)
  - lokales Dateisystem: ZFS
  - Netzwerkdateisystem: NFS (v3) und samba
  - *Hot spare Replikation* durch AVS

Details bestehendes Setup (Typ A)

- clients ($\approx$ 50): Linux-Laptops, Datensync per unison
- Login Server (2): remote Desktop, ssh, Datentransfer
- Compute Server (6): remote Desktop, ssh, Datentransfer, FHGFS, WiRe, CUDA
- Fileserver (2): (kein SAN)
  - lokales Dateisystem: ZFS
  - Netzwerkdateisystem: NFS (v3) und samba
  - *Hot spare Replikation* durch AVS
- Sonstige Server (6): Virtualisiert mit OpenVZ
  - Email & Groupware und LDAP
  - Diverse Lizenzserver
  - Nameserver und dhcpd
  - puppet und FAI

Schwierigkeiten des Setups

- Fileserver
  - ZFS (nur lokal)
  - NFS hat nur einen Zugangsknoten⇒ ineffiziente Hardware-Nutzung, keine intrinsische Redundanz
  - NFS wegen Sicherheit und des clientssetups nur intern nutzbar
  - **AVS** verursacht Last auf dem Haupt-Server

Schwierigkeiten des Setups

- Fileserver
  - ZFS (nur lokal)
  - NFS hat nur einen Zugangsknoten⇒ ineffiziente Hardware-Nutzung, keine intrinsische Redundanz
  - NFS wegen Sicherheit und des clientssetups nur intern nutzbar
  - **AVS** verursacht Last auf dem Haupt-Server
- Sonstige Server
  - Unbenutzte Speicherkapazität
  - Unbenutzte Rechenkapazität (Balancing nur schwer möglich)
  - Unzureichende Verfügbarkeit bei Fehlern (single point of failure)

Schwierigkeiten des Setups

- Fileserver
  - ZFS (nur lokal)
  - NFS hat nur einen Zugangsknoten⇒ ineffiziente Hardware-Nutzung, keine intrinsische Redundanz
  - NFS wegen Sicherheit und des clientssetups nur intern nutzbar
  - **AVS** verursacht Last auf dem Haupt-Server
- Sonstige Server
  - Unbenutzte Speicherkapazität
  - Unbenutzte Rechenkapazität (Balancing nur schwer möglich)
  - Unzureichende Verfügbarkeit bei Fehlern (single point of failure)
- Clients
  - Nicht sehr mobil (Mobilität erfordert Restart)
  - Datensynchronisation (Unison) kostet Zeit und Ressourcen bei Login/Logout
  - User muss Vorgang verstehen und beobachten.

Verbesserungsansatz

Serverseitig: Nutzung von verteilten, fehler-toleranten Dateisystem (Distributed parallel fault-tolerant file systems) und damit storage-cluster und VM-Cluster.
clientseitig:

- Home abkoppeln, nur einzelne Verzeichnisse mounten.
- DiscoFS . FS-Overlay mit vollständigen lokalen Cache (FUSE)
- (geplante) Ceph-erweiterung für disconnected operations

DiscoFS - DISCOnnected File System (optional)

(Von Robin Martinjak) https://github.com/rmartinjak/discofs

Idee

FS-Overlay über lokalen Daten und remote Daten, die in Sync gehalten werden sollen.

- disconnected operations: Bei disconnect lokal, bei connect lokal und remote.
- Conflict-Handling wenn sich beide Quellen geändert haben.

Komponenten:

- Cache: lokale Daten
- Sync table: Synctime of file objects
- Job: Replikation organisieren
- Database: Job und Sync-Daten speichern
- State check Thread: Online-Status prüfen.
- worker Thread: Jobs abarbeiten

Ceph-Erweiterung

Idee

Benutze CRUSH um eine zusätzliche Replica eines Homes lokal auf dem Laptop abzulegen.

- Geht die Verbindung verloren sind alle Daten lokal und, theoretisch, kann der lokale Ceph-client weiterarbeiten.
- Das Wiederverbinden ist bereits implementiert, allerdings bräuchte man noch ein Conflict-Handling.

Aufbau

Anforderungen an ein verteiltes Dateisystem

- Hohe Performanz durch Striping
  - Durchsatz
  - Latenz
- Skalierbarkeit
- Verfügbarkeit
- Zuverlässigkeit

Zuverlässigkeit/Verfügbarkeit

Problem: Fehler sind mehr die Norm als die Ausnahme

- System günstiger Hardware
- Fehler in Programmen
- Menschliches Versagen
- Betriebssystemfehler
- Ausfall von:
  - Festplatten
  - Arbeitsspeicher
  - Kabel
  - Netzwerk
  - ...

Moderner Ansatz

- Objektbasiert
- Festplatten $\rightarrow$ Intelligent object storage devices (OSD's)
  - CPU
  - Netzwerkschnittstelle
  - Lokaler Cache
- Metadaten getrennt von Nutzdaten

Aufbau

1. Einführung und Motivation

2. **Parallel, distributed, fault-tolerant filesystems**
   - Auswahl verteilte Dateisysteme
   - Ceph Theorie
   - Ceph Praxis

3. Erfahrungen

Auswahl verteilte Dateisysteme

Lustre Alter Kernel und nur ein Metadatenserver

Google FS Proprietär, ausgelegt auf große Dateien

GlusterFS Nur FUSE, keine advanced features

XtremeFS nett ;) aber Fokus liegt auf Geo-Features

FHGS Proprietärer server, Opensource client. keine Redundanz, im HPC-Umfeld aber gut (Bei uns seit kurzem im Einsatz).

pNFS Vielversprechend, bisher aber keine Tests durchgeführt (zu spät?)

Aufbau

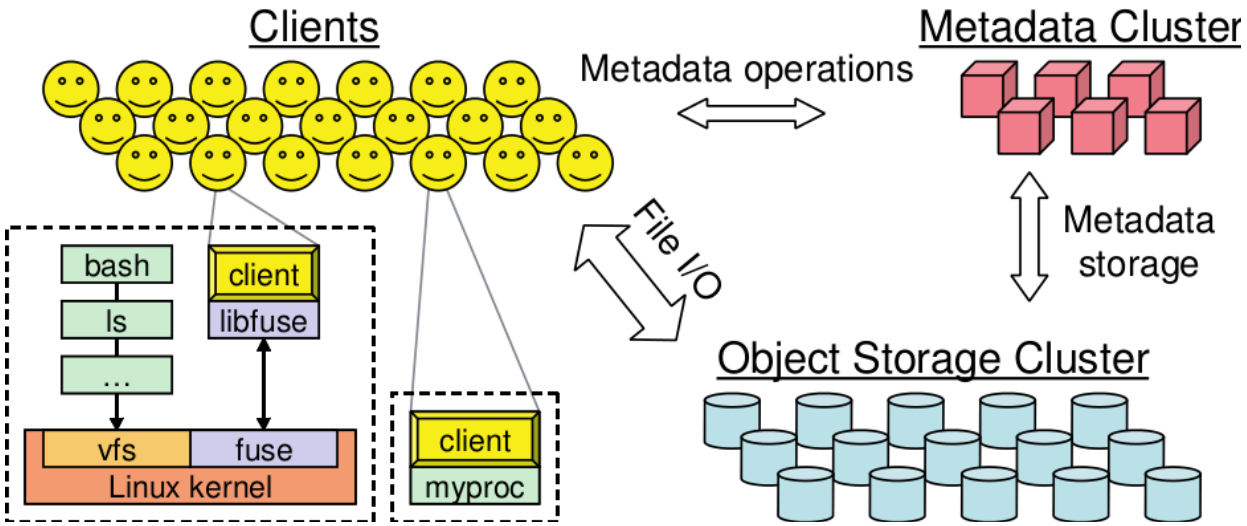
Ceph - Übersicht

Drei Komponenten

- Client: Posixähnliche Schnittstelle
- OSD Cluster: Speichert Daten und Metadaten
- Metadaten Cluster
  - Verwaltet Namensraum
  - Konsistenz/Kohärenz
  - Sicherheit (Noch nicht ausreichend implementiert)

Drei Zugriffsmöglichkeiten

- Object-Store (Swift, Amazon S3, RESTful, ...)
- Block-device (RBD)
- Filesystem (CephFS, POSIX, Snapshots)


Clients
Metadata operations
Metadata Cluster
File I/O
Metadata
storage
bash
ls
...
client
libfuse
vfs
fuse
Linux kernel
client
myproc
Object Storage Cluster

Quelle: [**?**]

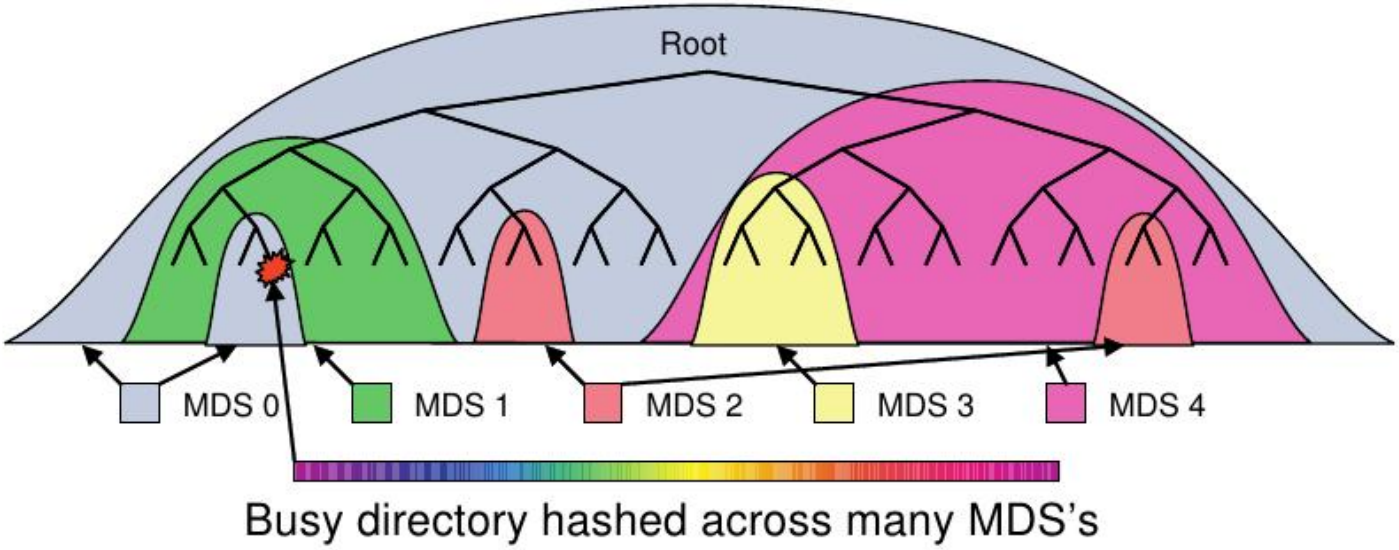
Design Features

- Trennung der Verwaltung von Daten und Metadaten
  - Unabhaengige Skalierbarkeit
  - Keine Block/Objektlisten (CRUSH)
- Dynamische Verteilung der Metadaten
  - Halber Aufwand sind Metadatenoperationen
  - Gutes Loadbalancing der Metadaten ist äußerst wichtig
- Autonomes OSD Cluster verantwortlich für:
  - Replikation
  - Fehlererkennung
  - Wiederherstellung nach Fehlern

Dateizugriff (CephFS)

1. Client sendet Dateirequest an das MDS Cluster
2. Ein MDS übersetzt Dateiname in *file inode*
   - Unique inode number
   - Besitzer
   - Größe
   - ...
3. MDS gibt inode number, Größe und Verteilungsstrategie zurück
4. Direkter Zugriff von Client auf OSD

Dynamische Metadaten-Verteilung



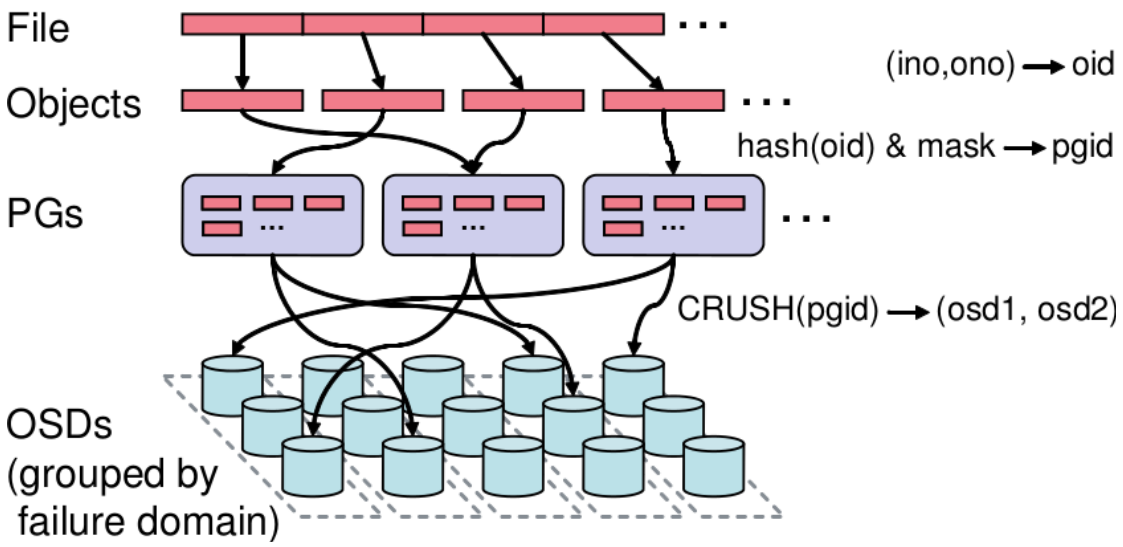
Quelle: [**?**]

Client Synchronisierung

Problem: Mehrere Clients schreiben/lesen eine Datei gleichzeitig.

- Kein read caching/write buffering mehr
- I/O der clients synchron
  - Synchrone I/O schlecht für Client Performanz
- HPC Erweiterung: O_LAZY flag für `open`

CRUSH


File
Objects
PGs
OSDs
(grouped by
failure domain)
(ino,ono) → oid
hash(oid) & mask → pgid
CRUSH(pgid) → (osd1, osd2)

CRUSH!

Für $x \in \mathbb{N}$ ist *CRUSH* eine Abbildung nach $R \in \mathbb{N}^n$ für $n \in \mathbb{N}$.
x : Data-Object
R : Placement für n Replicas

Aufbau

Vorraussetzungen

- Ausgeschaltetes Write Caching auf der Festplatte
- Dateisystem mit *Extended Attributes (XATTRs)*:
  - XFS
  - BTRFS
- Empfohlen: Eine Platte je OSD

Installation einfach, Pakete vorhanden für:

- DEB (Ubuntu/Debian)
- RPM (CentOS, RHEL)

Cluster Design

3 Daemons

- ceph-osd: Object Storage
- ceph-mds: Metadaten Server: Verteilter, kohärenter Cache für Metadaten
- ceph-mon: Monitor, Cluster Management, Konfiguration und Zugangspunkt für CephFS

Object Storage Daemon

- Mehrere OSDs pro Host möglich
- Hardware Empfehlung:
  - Viele Festplatten
  - Viel Ram (besseres Caching) (min. 500MB/Daemon)
  - Schnelles Netzwerk
- Anzahl:
  - Mindestens 2, so viele wie möglich

Metadaten Daemon

- Hardware Empfehlung:
  - Sehr viel Ram
  - Schnelle CPU
  - Schnelles Netzwerk (geringe Verzögerung)
- Anzahl:
  - Mindestens 1
  - 2 oder mehr für Redundanz und Load Balancing

Monitor Daemon

- Hardware Empfehlung:
  - Einige GB Festplattenspeicher
  - Feste IP
- Anzahl:
  - Ungerade Anzahl! (Split Brain)
  - 1 ist ok
  - 3 ideal für die meisten Anwendungsfälle
  - Mehr nur für sehr große Cluster

Beispiel ceph.conf

```
[global]
auth supported = cephx
osd pool default size = 2  # Write an object 2 times.
[osd]
osd journal size = 1000
[mon.a]
host = myserver01
mon addr = 10.0.0.101:6789
[mon.b]
host = myserver02
mon addr = 10.0.0.102:6789
[mon.c]
host = myserver03
mon addr = 10.0.0.103:6789
[osd.0]
host = myserver01
[osd.1]
host = myserver02
[osd.2]
host = myserver03
[mds.a]
host = myserver01
```

Deployment

- **ceph-deploy**
  1. Benötigt passwortlosen `root`-Login auf allen Hosts
  2. ceph-deploy new mon
  3. ceph-deploy osd create
  4. ceph-deploy mds create

CRUSH-map

```
# buckets
host c096-137 {
        id -6           # do not change unnecessarily
        # weight 2.000
        alg straw
        hash 0  # rjenkins1
        item osd.6 weight 1.000
        item osd.7 weight 1.000
}
...
rack unknownrack {
        id -3           # do not change unnecessarily
        # weight 6.000
        alg straw
        hash 0  # rjenkins1
        item c096-134 weight 2.000
        item c096-135 weight 2.000
        item c096-136 weight 2.000
        item c096-137 weight 2.000
}
root default {
        id -1           # do not change unnecessarily
        # weight 8.000
        alg straw
        hash 0  # rjenkins1
        item unknownrack weight 8.000
}
```

Aufbau

CephFS

Gut

- Selbstheilung sehr gut.
- Feature-Set sehr gut (insbesondere Zugriffsmöglichkeiten).

Schlecht

- Crash bei mount mit dem Kernel Treiber auf OSD-Server (per Design).
- Performance bei vielen kleinen Dateien/Datenbanken noch relativ gering (Hardware? Laut Entwicklern ist auch CephFS noch nicht ganz so ausgereift).

Alte Tests / alter Cluster

virt

Phoronix Test Suite	lxc-on-ext4	lxc-on-gluster	lxc-on-cephrbd-ext4	lxc-on-cephrbd-direct-ext4
Apache Benchmark	7747.08	1653.77	6721.80	7108.74
NGINX Benchmark	8322.34	584.71	7009.10	7818.15
PostgreSQL pgbench	227.18	41.52	6.60	11.31
PostMark	1383	114	1099	1154
IOzone	49.69	27.07	7.86	7.31
IOzone	1485.55	98.72	1141.47	1120.62
C-Ray	194.12	190.51	187.42	190.70
POV-Ray	1268	1280	1275	1304
PHPBench	31765	31692	31580	31791
GraphicsMagick	80	75	80	79
OpenSSL	41.30	41.55	41.63	41.60
Timed MAFFT Alignment	18.84	19.58	20.65	19.68
Himeno Benchmark	282.63	290.40	294.70	293.70
LAME MP3 Encoding	38.83	39.15	39.05	39.17
LZMA Compression	270.05	303.19	268.22	264.46
Stream	5468.09	4637.33	5268.77	5503.62
Stream	5373.22	4548.70	5132.91	5380.74
Stream	5791.56	4958.28	5415.82	5883.06
Stream	5867.09	5034.31	5481.42	5959.16

PHORONIX-TEST-SUITE.COM

Benchmark: `https://rcs.num.math.uni-goettingen.de/virt-benchmark/composite.xml`

Vorläufige neue Tests

NFS : 1 Node, 12 Platten
NFS (MI): 1 Node, 8 Platten (neuer)
CephFS: 6 Nodes, 4 OSD-Nodes, 8 SAN-Platten, 2 Replica

Setup	read Kb/s	write Kb/s	read Kb/s (64)	write Kb/s (64)
NFS (MI)	78995	44613	8623	2162
NFS	316	164	790	223
CephFS	137613	18012	843	211

Zusammenfassung

Positiv

- Sehr vielversprechend und in aktiver Entwicklung.
- Team sehr offen und hilfsbereit.
- Fülle an Zugriffsmöglichkeiten und Skalierbarkeit.

Negativ

- Einiges unvollständig implementiert (Sicherheit)
- Insbesondere CephFS noch nicht voll Produktionsreif

Fragen? Anregungen? Vorschläge?

Literatur