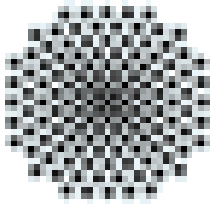


Hauptseminar Clustercomputing
I/O und verteilte Dateisysteme: Lustre und PVFS2

Markus Götz

Markus.Goetz@studi.informatik.uni-stuttgart.de

5.6.2007

Inhalt

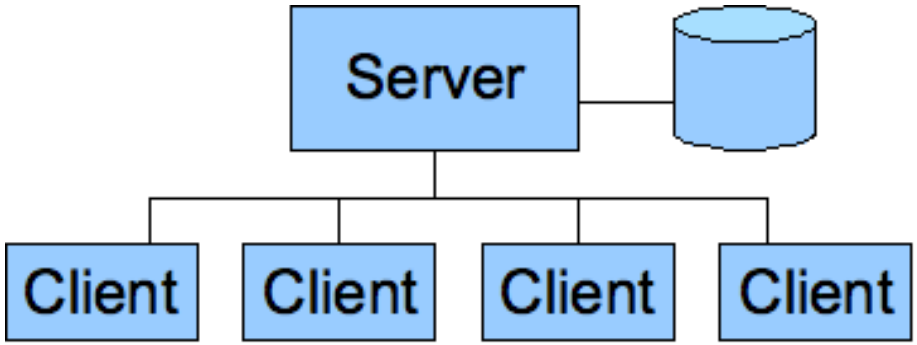
Beispiel: Paralleles Rendering und Encoding

- Hier: CGI Filme (=Computer-Generated-Imagery)
- In unabhängige Frames unterteilbar
- Eine Sekunde besteht aus meist 24 Frames
- Quellmaterial: Modelle, Texturen, Parameter, ...
- Ausgabe: Frame
- Frame weiter unterteilbar
- Parallelität bei Encoding

Ziele

- I/O = Input/Output
- Ziel: Entkopplung von Berechnungen und (Fest-)Speicher $\rightarrow$ getrennt optimierbar
- Gewünschte Eigenschaften
  - Einheitlicher Namensraum + Lokationstransparenz
  - Hohe Performanz
  - Große Dateien und Dateisysteme
  - Paralleler Zugriff
  - Datenkonsistenz
  - Fehlertoleranz
  - Skalierbarkeit

Warum kein NFS?



- nicht skalierbar
- keine hohe Performanz
- nicht optimiert auf parallelen Zugriff

Striping

- Festplatten: Hohe Zugriffszeit, mechanische Geräte, eingeschränkte Bandbreite, keine Parallelität
- Lösung: Eine Datei auf mehrere Festplatten verteilen
  - Verschiedene unabhängige kleine Requests können gleichzeitig beantwortet werden
  - Größere Requests über mehrere Blöcke können von verschiedenen Platten kommen
- Einzelne Stripes meist 64 bis 256 kB groß, Verteilung über die Server oft "round-robin"

Access-Patterns

- Programme auf Clustern wollen eine Datei nicht unbedingt sequentiell lesen/schreiben → noncontiguous access
- Beispielsweise:
  - Bestimmte Bytebereiche
  - Alle n Bytes m Bytes lesen (auch komplexere Patterns möglich)
- Explizite Unterstützung durch das Dateisystem bringt Performanzgewinn

Collective I/O

- Oft wollen mehrere Prozesse die Gleichen oder nah beieinander liegende Daten lesen/schreiben
  $\rightarrow$ Es macht Sinn, sich zusammenzutun
- Implementierung kann mehrere kleine I/O-Operationen in eine Große kombinieren

Caching/Buffering + Locking

- Caches/Puffer
  - Erhöhen die Performanz
  - Server-side vs. Client-side
- Probleme: Kohärenz, Sequentielle Konsistenz
- Eine Lösungsmöglichkeit ist Locking:
  - Gesamte Datei locken
  - Bereich locken
  - Block/Stripe locken

RDMA

- = Remote Direct Memory Access
- Speicher-zu-Speicher-Transfer zwischen zwei Rechnern ohne Hilfe durch das OS
- höherer Durchsatz, geringere Latenz
- Verschiedene Standards und Implementierungen

Metadaten

- Allgemein: Daten über Daten
- Hier: Dateinamen, Verzeichnis-Hierarchie, Datum der letzten Änderung, Zugriffsrechte, ...
- Essentiell zur Lokalisation meiner Dateien
- Man beachte den Größenunterschied von Daten zu Metadaten
- Ein Dateisystem lebt von seinen Metadaten!

Failover

- Übernahme durch anderen Knoten wenn ein Ausfall auftritt
- active/passive - Zwei Knoten, nur einer ist aktiv. Der Andere übernimmt bei einem Ausfall dessen Aufgaben und Daten
- active/active - Beide Knoten aktiv. Ein Knoten springt bei einem Ausfall für den Anderen ein
- Benötigt oft spezielle Hardware, z.B. Shared Storage, Heartbeat-Verbindung

Journaling

- Bei normalen Dateisystemen relevant $\longrightarrow$ diese werden jedoch oft intern von Clusterdateisystemen benutzt
- Metadaten-Journaling vs File Journaling
- Ermöglicht die Überführung von einem konsistenten Zustand in einen konsistenten Zustand
- Ermöglicht schnelles Recovery

Zusammenfassung

- Viele interessante Prinzipien
  - zur Erhöhung der Performanz
  - zur Steigerung der Zuverlässigkeit
- Teilweise Anwendung in GPFS, CXFS, Lustre und PVFS2

Lustre

- Einführung
- Zielgruppe
- Eigenschaften
- Architektur
- Zugriff
- Zusammenfassung

Einführende Informationen zu Lustre

- `http://www.lustre.org`
- Lustre = Linux + Clusters
- Dual-Lizensierung: GPL + Kommerziell
- 2003: Version 1.0, Hier: 1.4.x
- Für Linux 2.4 und 2.6 (serverseitig)
- Unterstützung durch US Behörden
- Läuft auf Standardhardware (. . . und mehr)
- Läuft über TCP/IP, Quadrics Elan, Myrinet, Mellanox, Infiniband
- Heterogene Rechner/Netze werden unterstützt

Lizensierung

- Kein Community-Produkt im üblichen Open-Source Sinn
- “designed, developed and maintained by Cluster File Systems, Inc.”
- GPL-Variante erscheint zur gleichen Zeit wie die Kommerzielle
- CFS, Inc hält “copyright to virtually 100% of the relevant Lustre source code”

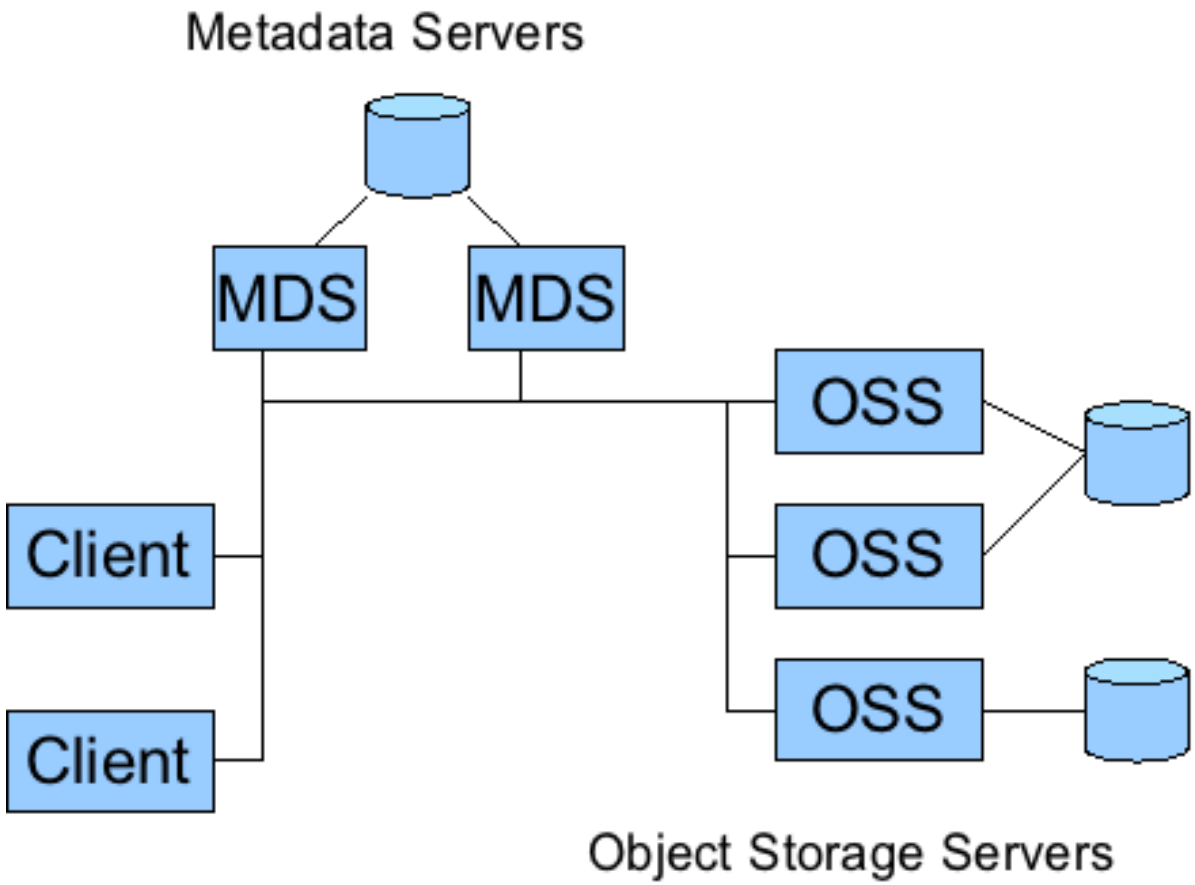
Zielgruppe

- Large-scale Applications ("clusters with 10,000's of nodes, petabytes of storage, move 100's of GB/sec“)
- Benutzt u.a. BlueGene/L (Platz 1 auf der Top500 Liste!)
- Site-Wide File Systems

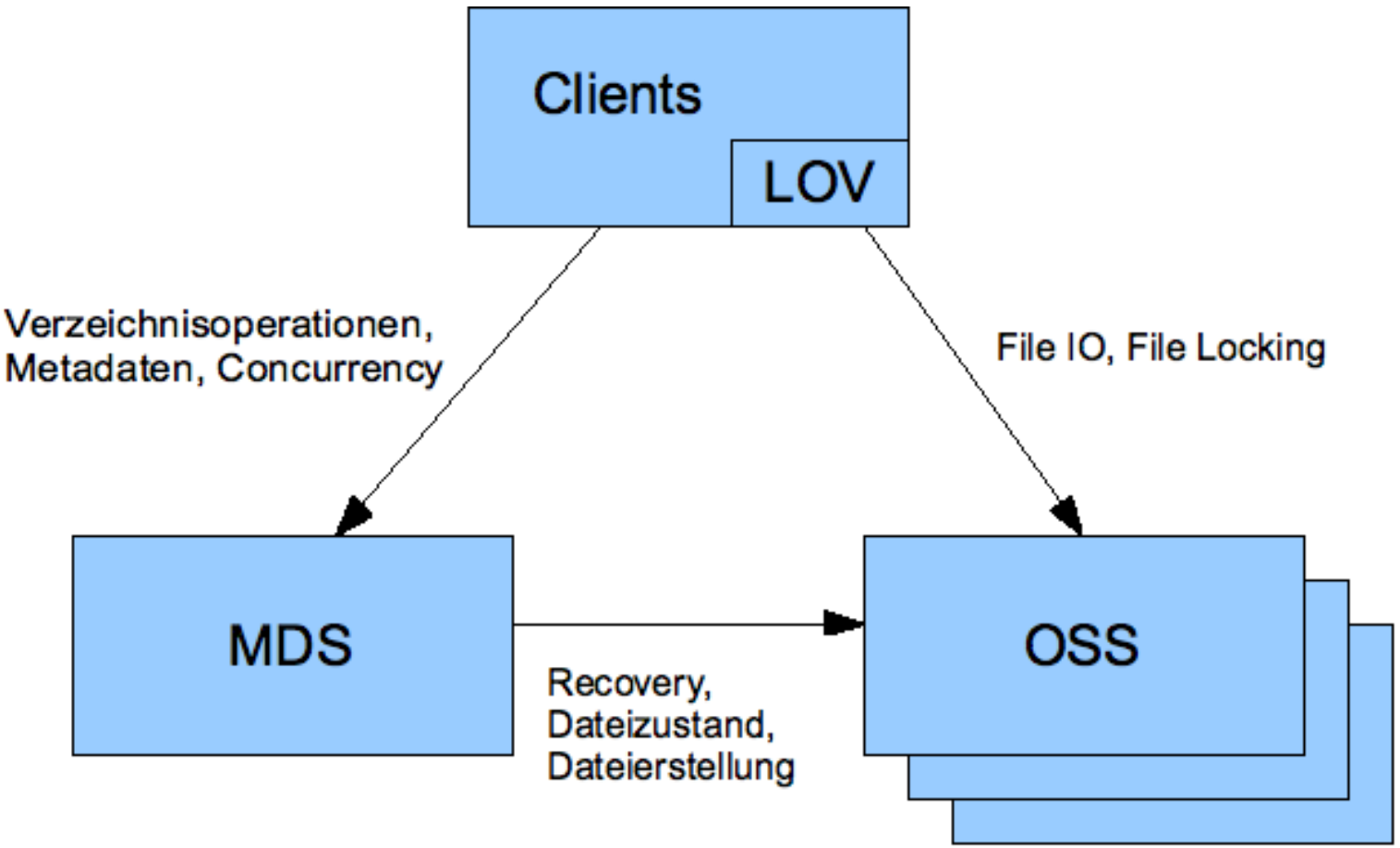
Eigenschaften

- Benutzt intern ein modifiziertes ext3 (Journaling usw!)
- POSIX-Semantiken beim Dateizugriff
- “All clients which mount the file system will see a single, coherent, synchronized namespace at all times.”
- Für beliebige Verzeichnisse und Daten geeignet
- Failover-Unterstützung
- LDAP-Server zur Unterstützung
- Properties: distributed, parallel, clustered, asymmetric, object-based, (fault-tolerant)

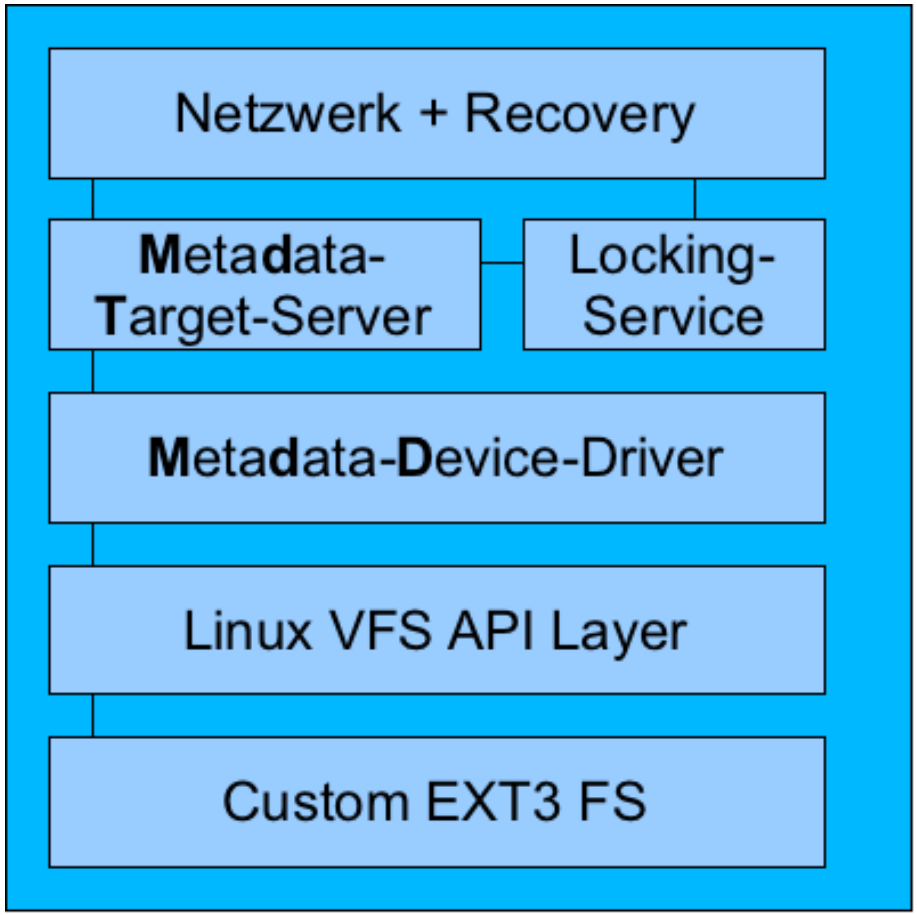
Architektur I



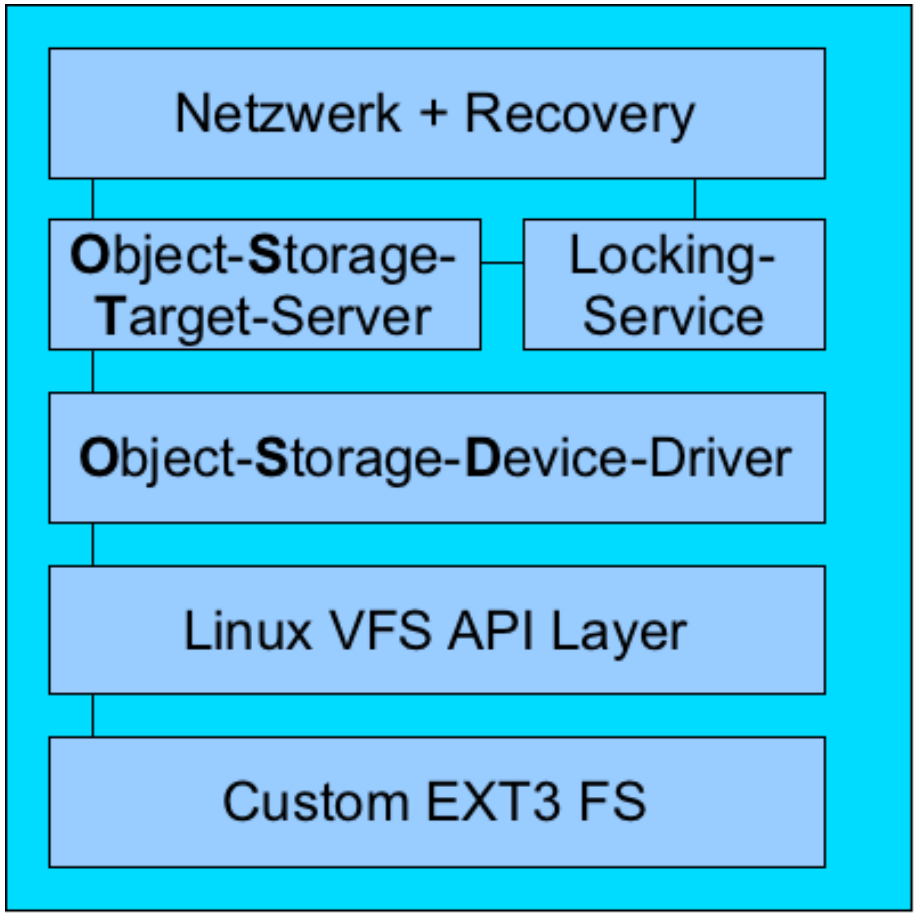
Architektur II



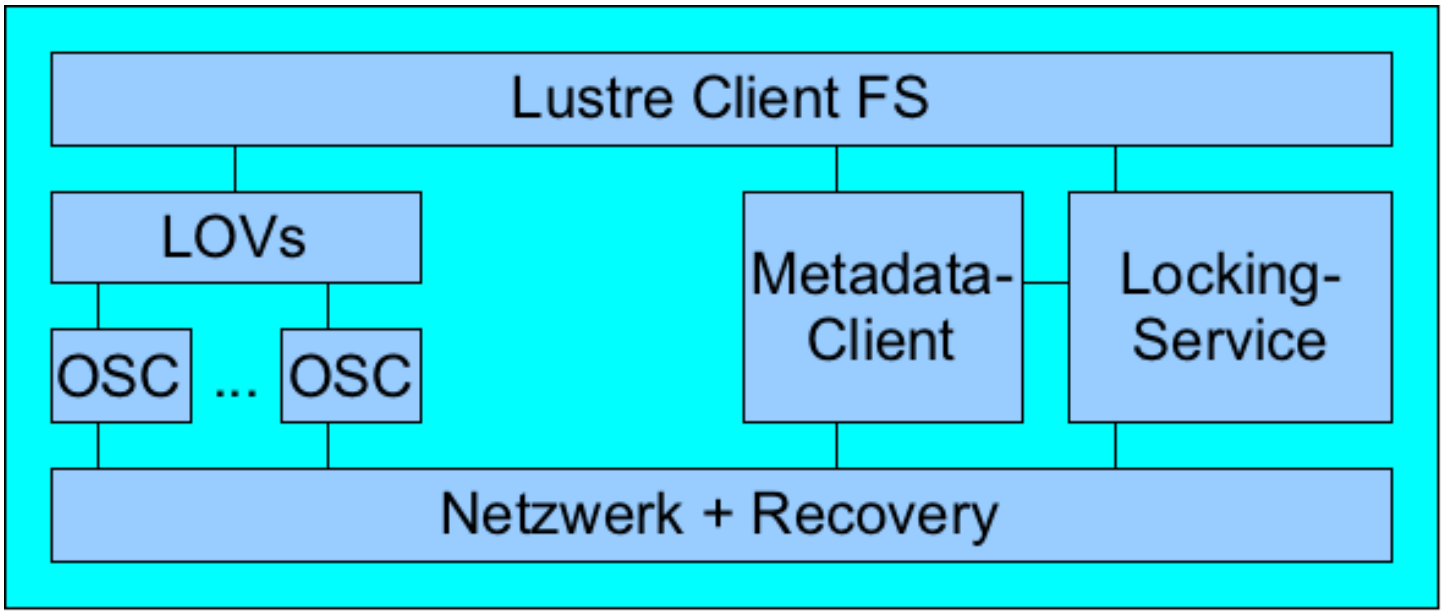
Architektur III - Metadata Server



Architektur IV - Object Storage Server



Architektur V - Clients



Architektur VI - Freispeicher / Striping

- Verteilung neuer Dateistripes basiert auf
  - Lastverteilung
  - Ausnutzung des freien Speichers
- Round-robin bei weniger als 20% Unterschied in der Größe des freien Speichers
- Ansonsten Zufallsverteilung gewichtet nach freiem Speicher

Zugriff durch Clients

Mehrere Möglichkeiten:

- Linux VFS
  - Mounten ins Dateisystem
- liblustre
  - Emuliert ein Mounten ins Dateisystem
- MPI-IO
  - Datenaustausch via Message-Passing ist möglich

liblustre

- Emuliert die Dateisystemzugriffe im Userspace
- Shared Library, Benutzung via `lrun` bzw `LD_PRELOAD`
  $\rightarrow$ Unmodifizierte Programme unter Linux
- verändert `open`, `read`, `write`, ...
- Gaukelt ein `/mnt/lustre` vor
- Ermöglicht Zugriff von Windows u.a.

Zusammenfassung Lustre

- Allround-Dateisystem für große Netze und Cluster
- Einsatz erfolgreich in der Praxis erprobt
- Interna nicht so gut dokumentiert wie man sich das als interessierter Student wünscht

PVFS2

- Einführung
- Zielgruppe
- Eigenschaften
- Architektur
- Zugriff
- Zusammenfassung

Einführung

- `http://www.pvfs.org`
- PVFS2 = Parallel Virtual File System 2
- PVFS2 Releases seit 2003, hier Version 2.6.3
- Für Linux und Andere
- Lizenz ist GPL. Jedoch keine Firma dahinter wie bei Lustre
- Unterstützung durch US Behörden/Universitäten
- PVFS1 = Vorgänger, wird nicht mehr intensiv weiterentwickelt
- Benötigt Standardhardware
- Läuft über TCP/IP, Myrinet und Infiniband
- Heterogene Hardware/Netze möglich

Zielgruppe

- Scratch-Dateisystem $\rightarrow$ Speicherplatz für große Berechnungen, nicht für Langzeitspeicherung
- D.h. PVFS2 ist kein Allround-Dateisystem für den "normalen" User
- Failover/Redundanz kein Entwicklungsziel, aber möglich

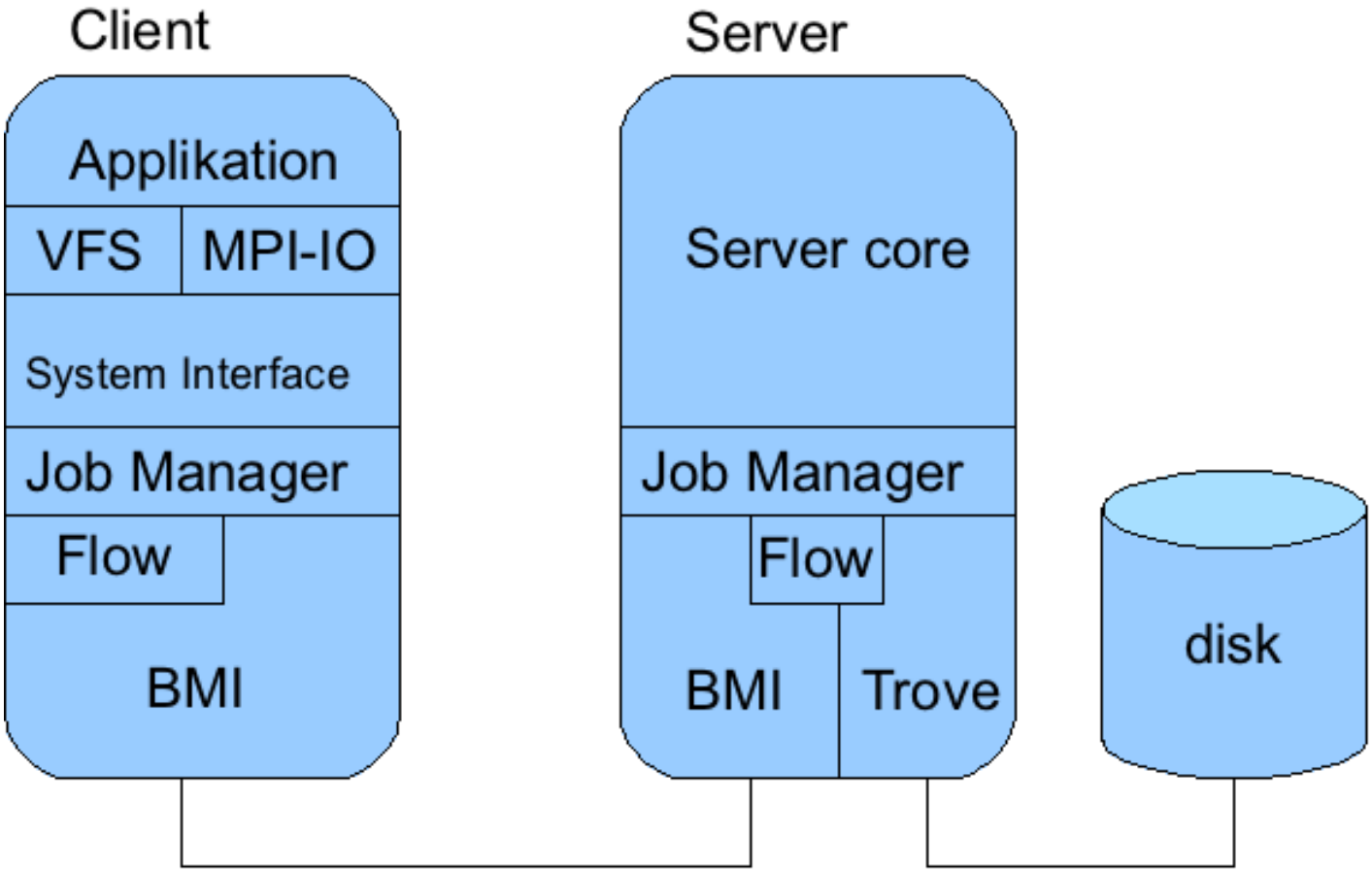
Eigenschaften

- virtuell → benutzt intern ext3, . . . (Journaling!)
- Zugriff erfolgt zustandslos (keine Locks, keine offenen Dateien) → vereinfachtes Recovery bei Fehlern
- keine Caches
- Insbesondere also keine POSIX-Semantik!
- Beides führt zu mehr Performanz, aber auch zu mehr Programmieraufwand
- Ins Dateisystem einhängbar oder via MPI-IO benutzbar
- Benuzerdefinierte Dateiverteilung (Striping, . . .)
- Failover-Unterstutzung
- Properties: distributed, parallel, clustered, asymmetric, object-based, (fault-tolerant)

Architektur I

- Hauptsächlich Standard-C-Programmcode der im Userspace ausgeführt wird
- `pvfs2-server`-Prozess für (konfigurierbar)
  - Metadaten
  - Dateien bzw. Teile davon
- `pvfs2-client`-Prozess für Clients

Architektur II



Zugriff durch Clients

Mehrere Möglichkeiten:

- Linux VFS
  - Mounten in das Dateisystem
- MPI-IO
  - Zugriff über Message-Passing

MPI-IO

- MPI = Message Passing Interface = Schnittstelle für den Nachrichtenaustausch zwischen Prozessen
- MPI-IO ermöglicht die Ein- und Ausgabe über MPI Nachrichten
- Lesen entspricht Empfangen, Schreiben entspricht Senden
- Flexibility, Portability, Interoperability
- Noncontiguous access
- Definierbare File View
- Blocking/Nonblocking
- Collective/Noncollective
- Hints
- Start über `mpirun`

MPI-IO Beispiel

```
int myrank, buf[BUFSIZE];
MPI_File thefile;
MPI_Offset offset;
...
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_File_open(MPI_COMM_WORLD, filename,
  (MPI_MODE_WRONLY | MPI_MODE_CREATE),
       MPI_INFO_NULL, &thefile);

MPI_File_set_view(thefile, 0, MPI_INT,
 MPI_INT,
 "native", MPI_INFO_NULL);

offset = myrank * BUFSIZE;
MPI_File_write_at(thefile, offset, buf,
 BUFSIZE, MPI_INT,
      MPI_STATUS_IGNORE);

MPI_File_close(&thefile);
...
```

Zusammenfassung PVFS2

- Scratch-Dateisystem für Cluster
- Einsatz erfolgreich in der Praxis erprobt
- Interna besser dokumentiert als bei Lustre

. . . im Vergleich

Lustre Skalierbares POSIX Dateisystem für große Datenmengen mit Unterstützung von paralleler Ein- und Ausgabe

PVFS2 Paralleles skalierbares Dateisystem für Spezialfälle

Danke für die Aufmerksamkeit! Gibt es noch Fragen?

Quellen

- `http://www.pvfs.org` (Quickstart, FAQ, Guide, Mailinglisten, ...)
- `http://www.lustre.org` und `http://clusterfs.com/` (Manual, Whitepaper, Wiki, Mailinglisten, ...)
- `irc://irc.freenode.net/pvfs2`
- `irc://irc.freenode.net/lustre`
- `http://de.wikipedia.org/wiki/Cluster-Dateisystem`
- `http://www.developers.net/intelisnshowcase/view/991`
- `http://www.clustermonkey.net/content/view/126/32/`
- `http://www.clustermonkey.net/content/view/31/32/1/1/`
- `http://www.pvfs.org/files/linuxworld-JAN2004-PVFS2.ps`
- Heiko Bauke, Stephan Mertens: Cluster Computing, Springer, Berlin, 2005
- John M. May: Parallel I/O for High Performance Computing, Morgan Kaufmann, 2000
- Rajkumar Buyya: High Performance Cluster Computing: Architectures and Systems, Prentice Hall PTR, 1999
- Robert W. Lucke: Building Clustered Linux Systems (HP Professional Series), Prentice Hall PTR, 2004