Technologien zur Datenspeicherung in Rechenzentren

Klaus Kusche, Jänner 2015

Inhalt

- Motivation, Ziel, Voraussetzungen
- Ziele und Anforderungen
- Basistechnologien
- Grundlösung
- Verbesserte Lösung
- Die “Arme-Leute-Lösung” in Linux
- Alternative Lösung für Datenbanken

Motivation (1)

- Enorme *praktische Bedeutung* des Themas

  ==› Im Beruf mit hoher Wahrscheinlichkeit relevant
- Einige *interessante Ansätze und Technologien*

  ==› Erweitert die “informatische Allgemeinbildung”
- Kommt an Hochschulen meist viel zu kurz

  ==› Vermutlich “*Neuwert*”
- Mein persönlicher Background
  - In zwei Jobs beruflich damit befasst
  - Linux

Motivation (2)

Höchste Priorität einer Firma / einer Behörde /
eines Rechenzentrums- / Cloud-Dienstleisters ist der

unbedingte Erhalt
des <u>aktuellen</u> Datenbestandes!

(Rücksetzen auf das tägliche Backup ist <u>keine</u> Option!)

Datenverlust = ev. Firmenpleite:

- Betriebsstillstand, ev. manuelle Neuerfassung
- Finanzieller Verlust in Millionenhöhe
- Rechtliche Inanspruchnahme
- Ruf schwer geschädigt, Kundenverlust, ...

Motivation (3)

Zweithöchste Priorität einer Firma / einer Behörde /
eines Rechenzentrums- / Cloud-Dienstleisters ist die

Minimierung der Stillstands-Zeiten!

(geplante und Ausfall-bedingte)

Folgen wie zuvor: Kosten, Rufschädigung, ...

Heute meist ***"Service Level Agreement"***:

Vertrag legt Verfügbarkeit, Ausfallzeiten, ... genau fest

==> Bei Nicht-Erfüllung: *Kein Geld!*

Motivation (4)

Der Enterprise-Storage-Markt

- ist ein ***Milliarden-Markt***
  (Firmen wie EMC, HDS, ... leben nur davon, und das gut!)
- hat sehr ***gesunde Erträge***
  (bei Hardware, Software und Consulting)

Zur Orientierung:

Enterprise-Storage (HW & SW) kostet
pro Netto-Gigabyte weit mehr als
das <u>Hundertfache</u> einer nackten PC-Platte!

Ziel des Vortrags

- Verständnis der *Anforderungen und Ziele*
- Kenntnis typischer *Lösungskonzepte* und der dafür erforderlichen *Technologien*
- Kenntnis der *Vor- und Nachteile* dieser Konzepte

➔ *“Mitreden” können*

➔ *Lösungen einschätzen können*

Voraussetzungen

Wenig!

Grundkenntnisse von

- **Betriebssystemen** (insbes. Filesysteme, RAID)
- **Netzwerktechnik**
- **Hardware** (Plattenzugriff)

Ziele (1)

Schutz gegen

- **Desaster** (Brand, Hochwasser, Attentat, ...)
- **Infrastruktur-Ausfälle** (Netzwerk, Strom, Klima)
- **Alle Komponenten-Defekte** (Server, Controller, einzelne Platten, einzelne Netzverbindungen, ...)
- **Fatale "Soft-Errors"** (Abstürze, Systemstops z. B. wegen Double-Bit-RAM-Error)
- **Geplante Abschaltungen** (Verkabelungsarbeiten, Wartung, HW-Tausch und -Aufrüstung)

Anwendung muss weiterlaufen / verfügbar bleiben!

Ziele (2)

Kein *Schutz gegen*

- **Benutzerfehler** (falsche Dateneingabe / Löschung)
- **Software-Fehler**, die falsche Daten schreiben

In diesen Fällen ist
das Backup zuständig!!!

Keine hundertprozentige Verfügbarkeit
eines ***einzelnen*** Servers (z.B. durch gespiegelte Prozessoren)

Serverwechsel und
Unterbrechungen im (Sub-) Sekundenbereich
sind zulässig!

Ziele (3)

Typische Inhalte von Enterprise-Storage-Systemen sind u.a.:

- **Datenbank-Daten**
- **Images virtueller Maschinen**
- **Filesysteme mit Anwendungsdaten**

Nicht:

- **Systemplatte, Applikationscode, ...**
  (sind meist auf *normalen lokalen RAID1-Platten*)

==> Storage muss *nicht bootfähig* sein!

==> Server muss *ohne Storage hochfahren* können,
Daten-Storage wird erst *nachträglich gemountet*!

Die “Nicht-Lösung”

Netzwerk-Filesysteme (SMB/CIFS, NFS, AFS, ...):

- Verlagern den ***“Single Point of Failure”***
  nur vom Applikationsserver zum *Fileserver*

  ==> *Nichts gewonnen!*

- Erlauben *keinen direkten Plattenzugriff*
  (“raw” bzw. block-basiert)

  ==> Datenbanken “verweigern” Netzwerk-Files meist

- Fileserver (und das Netz dorthin)
  sind oft ***Bottlenecks***

- ...

Anforderungen (1)

- **Geographisch verteilt**

  ==> mindestens *2 Standorte* mit *mehreren km Abstand*

- **Redundante (“*gespiegelte*”) Daten**

  ==> *Jeder* Standort speichert *alle Daten*

  ==> … und zwar *lokal nochmals redundant* (RAID 5 / 10)

  Grund:
  Auch im Notbetrieb (nur 1 Standort)
  müssen die Daten redundant sein!

Anforderungen (2)

- **Redundante Hardware**

  ==› Ausreichend *Server* (für Vollbetrieb!)
  an *beiden* Standorten

  ==› Alle *Komponenten* (Platten-Controller,
  Netzwerk-Verbindungen und -Switches, ...)
  müssen *überall doppelt ausgelegt* sein!

- **Synchron**

  ==› Write-Befehle werden erst quittiert, wenn sie
  an *allen* verfügbaren Standorten *geschrieben* sind

  (d.h. jedes Write wirkt "*sofort*" überall)

Anforderungen (3)

- **Disk-Block-basiert, nicht File-basiert**

  ==› Storage wird aus System-Sicht angesprochen

  wie eine Platte,
  nicht wie ein Fileserver!

  Gründe u.a.:

  - Datenbanken, VM's usw. möchten u.U. *Raw-Zugriff*
  - *Filesystem-Features* sollen nutzbar bleiben
    (Snapshots und Versionierung, Live-Backups, ...)
  - Bei Ausfällen *robuster*:
    Volle Kontrolle was geschrieben ist und was nicht!

Erforderliche Basistechnologie

Direkter Zugriff (wie auf lokales Plattenlaufwerk)
auf *nichtlokales* Plattenlaufwerk oder *Storage-Subystem*:

- **Von mehreren Servern**
- **Über weite Distanzen**
- **Redundant (doppelt ausgelegt)**

==> Netzwerk-Technologie erforderlich!!!

- ***FC (Fibre Channel)***
- ***iSCSI***

FC Fibre Channel (1)

Seit rund 2 Jahrzehnten:

Standard-Technologie für SAN's
("Storage Area Network")

Verbindet:

- **Server <=> Storage-Subsystem**

  Storage-Subsystem =
  "Viele Platten mit sehr gutem lokalem RAID-Controller"

- ev. **Server <=> Bandroboter** (für Backup)

FC Fibre Channel (2)

- **Genormt, Hersteller-unabhängig**
  Controller für viele Server und Betriebssysteme verfügbar
- **Eigene HW, Glasfaser-basiert** (kurze Distanzen: *Auch elektrisch*)
  ==> Für große Entfernungen (bis 10 km)
  ==> Schnell (16 Gbit/s)
- **Eigenes Protokoll** (*nicht* TCP/IP-basiert)
  ==> Storage-optimiert, SCSI-kompatibel
  ==> Effizient, weniger Overhead als TCP/IP
- **Erlaubt redundante Topologien:**
  - FC-P2P: Point-To-Point (auch doppelt)
  - FC-SW: Geswitched (auch mehrstufig und redundant)
  - FC-AL: Logischer Ring (auch mit Bypass defekter Ports)
- "**IP over FC**", "**FC over IP**" und "**FC over Ethernet**" sind möglich

iSCSI (internet SCSI)

iSCSI = Ein Netzwerk-Protokoll zur Einbettung von SCSI in normales TCP/IP

(SCSI = Protokoll für lokalen Platten- und Bandlaufwerks-Zugriff)

==> Ermöglicht *blockweisen Direktzugriff*
von beliebigen Servern
auf beliebig viele Storage-Subsysteme
über *normale, bestehende TCP/IP-Netze*

- *NAS ("Network Attached Storage") = "Low Cost SAN"*:
  Viel **billiger** als FC *(FC ist $$$!)*
- **Ineffizienter:** Langsamer, mehr CPU- & Netz-Overhead
- Automat. TCP/IP-Rekonfiguration bei **Ausfall dauert**!

Reicht das? (1)

Hardware-mäßig ja:

- Min. 2 Server, min. 2 Plattensysteme
- *Doppeltes, kreuzweises* FC- oder iSCSI-Netz dazwischen

Spiegelung zwischen den Standorten:

- Entweder durch das *Storage-System*:
  Server schickt die Daten nur an ein Storage-System,
  dieses repliziert sie autonom auf das andere
- Oder durch den *Server* selbst:
  Server schickt die Daten *getrennt an beide* Standorte

Reicht das? (2)

Betriebssystem-mäßig jein:

Ohne spezielle Filesysteme
kann jedes Volume (logische Laufwerk)
des Storage-Systems zu jedem Zeitpunkt
nur auf einem Server gemountet sein!

Normale Filesysteme (NTFS, ext4, XFS, ...) können nicht von mehreren Servern gleichzeitig *verwaltet* werden

==> Jeder Server hat seinen *lokalen Cache* und ändert die *Metadaten* (Platzbelegung, Directories, ...) *für sich*

==> Das Filesystem würde dadurch *sofort korrupt*!

Grundlösung (1)

- Storage-*Volumes pro Anwendung* einrichten
- Jede *Anwendung* läuft zu jedem Zeitpunkt *nur auf **einem** der Server*
- Nur dieser Server hat *deren Volumes exklusiv gemountet*
- Der *andere Server* läuft entweder *leer* oder führt *andere Anwendungen* aus (auf *anderen* Volumes)

Grundlösung (2)

- **Im Störfall (Ausfall eines Servers):**

  Anderer Server *mountet* dessen Volumes
  und *startet* dessen Anwendungen

- **Im Störfall (Ausfall eines Storage-Systems):**
  - Server arbeitet *nur* mit dem
    anderen Storage-System weiter
    (deshalb muss jedes Storage-System
    intern die Daten nochmals als RAID speichern!)
  - Wenn defektes Storage-System *wieder online* ist:
    Nachziehen der Spiegelung (aller Writes)
    durch Server oder autonom durch Storage-System

Grundlösung (3)

Beurteilung:

- Relativ *einfach*, vielfach *erprobt*
- *Wenig Zusatzsoftware*, keine speziellen Filesysteme
- Umschaltvorgang ist durch Skripte *automatisierbar*
- *Filesystem-Zugriff* ist so *effizient* wie bisher

- *Hardware-Ausnutzung* ist *ineffizient*,
  Anwendungsdurchsatz ist durch einen Server limitiert
- *Umschalt-Vorgang dauert* ein paar Sekunden (ev. `fsck`)
- Clients müssen sich u.U. *frisch anmelden*

Gefahren (1)

- ***"Split Brain":***

  Beide Seiten sehen den jeweils anderen Server
  und das gegenüber liegende Storage-System nicht

  ==> *Beide starten* die Anwendung
  mit jeweils *nur ihrem* der beiden Storage-Systeme

  ==> Daten sind zwar *lokal konsistent*,
  aber zwischen den Standorten *unterschiedlich*

  ==> Daten eines Standortes müssen beim Zusammenführen meist verworfen werden!!!

Gefahren (2)

- Beide Seiten sehen den jeweils anderen Server nicht, aber sehen *beide* Storage-Systeme

  ==> *Beide mounten und starten* die Anwendung
  mit *beiden* Storage-Systemen

  ==> *Beide* Kopien der Filesysteme werden zerstört!
  (meist Rücksetzen auf Backup notwendig...)

Verschiedene Lösungsansätze, z.B. Linux-Brutal-Lösung:
Der Aktiv-Server dreht dem Passiv-Server
bei Abbruch der Server-Verbindung
z.B. per Handy-gesteuerter Steckdose den Strom ab
(mit jeweils verschiedener Verzögerung)

Spezielle Anwendungen

Manche Anwendungen (vor allem ***Datenbanken***!)
können auf mehreren Servern *gleichzeitig laufen*
und den parallelen Datenzugriff
anwendungsintern *synchronisieren*

==> Plattenzugriff

- entweder über *Raw-Zugriffe* statt über Filesystem

  ==> ganz ohne Mount eines Filesystems

- oder *ohne Änderung der Filesystem-Metadaten*
  (keine neuen Files, keine Größenänderungen, ...)

  ==> je nach FS: Ev. gleichzeitiger Mount möglich

Cluster-Filesysteme (1)

Cluster-Filesysteme (Betriebssystem-Option) erlauben

gleichzeitige Mounts
desselben Filesystems
auf mehreren Servern
(auch read/write)

==> Beteiligte Server stimmen sich laufend *über Netz* ab

- über *Allokationen*, Directory- und *Metadaten-Updates*
- über *File Locking und konkurrierende Updates*
- über *Caching*

Cluster-Filesysteme (2)

Beispiele:

- *Linux:* **GFS2**, **OCFS2** (Oracle), **Ceph**, **Lustre**, **GlusterFS**, ...
- *Kommerziell:* **Veritas**, ...

Ziele:

- Entweder ***Hochverfügbarkeit*** (*"HA"*: hier)
- Oder ***Skalierbarkeit*** (*"HPC"*: Supercomputing):
  Tausende Knoten, TB/s Durchsatz, PB Storage, ...

Anforderungen für Hochverfügbarkeit:

- Architektur *ohne* zentralen Verwaltungs-
  oder Metadaten-Server (wäre *Single Point of Failure*)

Cluster-Filesysteme (3)

Probleme:

- Im Vergleich zu normalen FS komplex und *ineffizient* durch *hohen Synchronisations-Overhead*
  (aber meist besser als Netzwerk-Filesysteme)
- Problem der *Abstimmung*,
  wenn sich die beiden Server nicht mehr sehen
  ==> meist vorsichtshalber *unmount auf **beiden** Servern*!

  Lösung: Cluster mit *mindestens 3* oder mehr Servern
  ==> wer *mehr als 50 %* Server sieht, darf aktiv bleiben
- Installation meist *nicht trivial*

==> *Wenn einfache Lösung reicht: Nimm einfache Lösung!*

Verbesserte Lösung

Mit Cluster-Filesystemen:

Mehrere Server können *gleichzeitig* dieselben Anwendungsdaten mounten & die Anwendung *ausführen* (wenn die Anwendung dafür geeignet ist)

==> Bessere *HW-Ausnutzung*, mehr *Durchsatz*,
ev. automatische *Lastverteilung*

==> *Schnelleres*, ev. unterbrechungsfreies *Failover*
(kein mount / `fsck`, kein Anwendungsstart)

Cluster-Filesysteme sind für Anwendungen normalerweise *transparent*
(verhalten sich aus Anwendungs-Sicht wie normale Filesysteme)
==> Anwendungen laufen unverändert

Poor Man's Enterprise Storage (1)

Erkenntnisse:

- Normale, *lokale Platten* sind mit Abstand am *billigsten*
  (und am *schnellsten*, vor allem beim Lesen!)
- Gute *TCP/IP-Netzwerk-Infrastruktur* ist oft vorhanden
- Server haben selten CPU-Last-Probleme,
  könnten effizient Spiegelung usw. in *Software* machen
- Betriebszustand ist meist
  zwei Server + zwei Plattensysteme,
  oder eins + eins im Notfall,
  aber selten zwei + eins oder eins + zwei

Poor Man's Enterprise Storage (2)

Ansatz:

- Keine getrennten Platten / Storage-Systeme

  ==› Platten an jedem Standort
  lokal im jeweiligen Server

- Jeder Server *spiegelt softwaremäßig*
  über normales LAN auf zweiten Server
  (d.h. eine RAID1-Hälfte lokale Platten,
  zweite RAID1-Hälfte Platten im anderen Server)

Erfordert "*Verlängerung*" der Plattenzugriffe über TCP/IP

Ginge im Prinzip mit iSCSI, aber ...

Linux DRBD (1)

DRBD =

"Distributed Replicated Block Devices"

- *Österreichische* Entwicklung (Fa. Linbit GmbH, Wien)
- GPL, im *Standard-Linux-Kernel*
- Viele aktive Installationen, viel Info

Umfasst auch:

- Automatische *Umschaltung*
- Automatische *Rück-Synchronisation* *(nur der Änderungen)*

Linux DRBD (2)

Ursprünglich:

Nur *aktiv/passiv*-Konfiguration:

- Nur jeweils *ein* Server darf ein Filesystem mounten
- Das Device bzw. Filesystem
  ist am sekundären (passiven) Server *unsichtbar*

Aber:

Andere Filesysteme können gleichzeigig
mit vertauschten Rollen gemountet werden
(jeder Server hat ein paar aktive und ein paar passive)

==> Deckt unsere "*Grundlösung*" voll ab

Linux DRBD (3)

Heute:

- Auch *aktiv/aktiv-Konfigurationen* möglich:

  Mount des Filesystems auf *beiden* Servern
- Weiters ist *aktiv/aktiv/passiv* möglich.

DRBD funktioniert an sich
mit *jedem Filesystem* oder auch *als Raw Device*,

aber für *aktiv/aktiv* ist ein *Cluster-Filesystem* nötig!

==> Entspricht unserer "*verbesserten Lösung*"

Datenbanken: Log Shipping (1)

Datenbank-Log = Dateien mit binärem, sequentiellem Protokoll aller durchgeführten DB-Operationen

Hauptzweck des Logs: Desaster-Recovery

Alter DB-Daten-Bestand (= Backup)

+

alle seitdem geschriebenen Logs

=

aktueller DB-Daten-Bestand

Datenbanken: Log Shipping (2)

Vorgehensweise:

- Zwei *völlig getrennte* Datenbank-Installationen (getrennte Server, getrennte Platten)
- Primäre (aktive) DB arbeitet normal
  + *schickt ihre Logs* laufend an sekundäre (passive) DB
- Sekundäre DB läuft im *Recovery Mode*,
  spielt Logs laufend in ihren eigenen Datenbestand ein

==> Operationen auf primärer DB
werden laufend auf sekundärer DB *nachgezogen*

Datenbanken: Log Shipping (3)

Unterstützung: *MS SQL Server, PostgreSQL, ...*

Eigenschaften:

- *Sehr einfach, sehr billig*:
  Nur Server mit *lokalen Platten* + normales TCP/IP-Netz
  Keine Remote Storage, keine Cluster-Software, ...
- Erhält auch bei Absturz *ACID-Eigenschaften* der DB

- Aber *hinkt einige Sekunden / Minuten* nach...
- und funktioniert *nur aktiv/passiv*!

“The end”

Fragen?