Open Source High Availability

Dr. Andreas Müller

afm@othello.ch

Slide 1

CH/Open Workshop, 14. September 2000

Agenda 1. Teil

1. Theoretische Grundlagen (Linux Conference Paper)
2. Test-Umgebung
3. Beispiel: `telnet`
4. Konfiguration für IP Address Failover
5. Demonstration verschiedener Szenarien:
   (a) Geplanter Failover und Recovery
   (b) Stromausfall und Wiederanlauf
   (c) Split Brain

Slide 2

Agenda 2. Teil

Slide 3

5. Fallstudien:
   (a) DNS
   (b) Mail-Gateway
   (c) Web-Server
   (d) File- oder Mail-Server
   (e) Datenbank-Server (Beispiel MySQL)
   (f) Firewall
   (g) Applikations-Server (Beispiele Zope und Apache Jserv)
6. Verschiedenes (Sicherheit, Erweiterungen)
7. Zusammenfassung

Test-Umgebung: Probleme

Slide 4

1. Box Count: $2\times$ für Cluster, $1\times$ als Client
2. Power Off! Filesysteme und Datenbanken können beschädigt werden.
3. Installation/Konfiguration: Vor Verlust schützen.

1. Um eine Failover-Situation testen zu können, braucht es mindestens drei Boxen: zwei Clusterkomponenten und einen Testrechner
2. Zu Testzwecken will man beide Clusterkomponenten jederzeit einfach ausschalten können, um einen Crash zu simulieren. Daran haben keine Freude:
   (a) Filesysteme: Inkonsistenz, wo sind die Daten geblieben?
   (b) Datenbanken: Nach einem Crash muss auch eine Datenbank Recovery fahren

   Vor jedem Crash: `sync;sync;sync`
3. Installation und Konfiguration: im schlimmsten Fall könnte durch den Crash die mühsam aufgebaute Konfiguration, und eventuell gepatchte Versionen der Software verloren gehen

Test-Umgebung

Slide 5

1. Diskless Systeme können keine Filesystemschäden haben (alle `writes` sind synchron)
2. bei Solaris 7 `logging` Option einsetzen, Linux mit ReiserFS installieren (noch nicht Standardkomponente des Kernels)
3. Konfiguration mit CVS auf einen "externen" Server sichern
4. Software von einem NFS-Server installieren
5. Maschinen simulieren: VMware
6. Gemischte Konfigurationen sind möglich (Solaris – Linux), solange keine Hardware gemeinsam genutzt werden soll

7. Gemeinsame Disks unter Linux problematisch

Slide 6

Demonstrations-Setup

1. Zwei x86-Maschinen mit Red Hat Linux 6.2, gemeinsames Ethernet
2. Notebook als Client
3. Untersuchter Dienst: `telnet`

Slide 7

4. Failover-Szenarien:
   (a) geplanter Failover und Recovery (Maintenance)
   (b) Crash (Stromausfall)
   (c) Split Brain

Dienstanalyse: Telnet

Slide 8

1. Dienst ist nicht an IP-Adresse gebunden (Port `*.23`)
2. Sitzungszustand:
   (a) TCP-Session: nicht failover-fähig, Benutzer beobachtet Zusammenbruch der Verbindung
   (b) Shell Environment: aus `.profile` etc. wiederherstellbar (Rollback auf Beginn der Session)
   (c) Backround-Jobs: müssen manuell wiederhergestellt werden
3. Aktion des Benutzers in jedem Fall erforderlich (kein automatischer Reconnect)
4. Im Login-Banner sieht man sofort, welche Maschine man erwischt hat.

Da nach dem Failover die TCP-Pakete an die Backup-Maschine geleitet werden, auf der der TCP Session Zustand nicht vorhanden ist, antwortet diese mit einem TCP Reset Segment. Beim Klienten wird die Fehlermeldung `connection reset by peer` erscheinen.

Konfiguration Dienst 152.96.232.12

Slide 9

1. Start-Script für den Dienst `152.96.232.12` mit verschiedenen Targets:
   (a) `start`: Dienst in den `RECOVER`-Zustand bringen, `failsh` wird den Rest machen.
   (b) `stop`: keine Aktion
   (c) `fail`: Dienst in den `FAIL`-Zustand bringen (Slave übernimmt).
   (d) `recover`: Dienst in den `RECOVER`-Zustand bringen (Slave gibt den Dienst auf, Master übernimmt)
   (e) `up`: Dienst starten (wird von `failsh` aufgerufen)
   (f) `down`: Dienst stoppen (wird von `failsh` aufgerufen)
   (g) `reclaim`: Recovery nach Split-Brain

Konfiguration Details

Slide 10

1. Demonstrations-Konfiguration für Linux
2. Slides zeigen Solaris-Konfiguration (abweichende Pfade):
   (a) in Solaris muss jedes Interface erst geplumbt werden (entfällt in Linux)
   (b) Linux verwendet eine Shellfunktion für den gratuitous ARP

Start-Aktion

Slide 11

```
# The start argument brings the service into a well defined
# recoverable state. If this is a master, it will later be
# started by the up method
/sbin/ifconfig ${interface}${subif} plumb
/sbin/ifconfig ${interface}${subif} inet ${servicename} \
        netmask ${netmask} broadcast ${broadcast} down
# for the next step to work, faild must be fully up, it
# may be necessary to include a small pause to give faild
# another chance
/opt/AFMfail/bin/failc -u -T -p ${port} -sRECOVER \
        ${servicename}
```

Resultat: definierter Zustand des Dienstes. Nicht verfügbar, aber jederzeit startbar

Up-Aktion

Slide 12

```
up)
        # starting the service without putting any information in
        # the faild
        /sbin/ifconfig ${interface}${subif} up
        # send gratuitous ARP
        /opt/AFMfail/sbin/grarp  ${interface} ${servicename}
        ;;
```

1. Wird nur von `failsh` aufgerufen, Registration in `faild` durch `failsh`.
2. **Resultat:** Dienst voll verfügbar

Fail-Aktion

```
fail)
        # The fail argument brings the service down, and marks it
        # as failed so that it will not be started again
        /sbin/ifconfig ${interface}${subif} down
        /opt/AFMfail/bin/failc -u -T -p ${port} -sFAIL ${servicename}
        ;;
```

Slide 13

1. Zum manuellen Stoppen des Dienstes (Tests, Maintenance)
2. **Resultate:** Dienst nicht mehr verfügbar, `failsh` über Zustandsänderung avisiert

Reclaim-Aktion

```
reclaim)
        # a slave has gone up although should not have done that
        # now we have to fix the IP-MAC assignment again
        /opt/AFMfail/sbin/grarp  ${interface} ${servicename}
        ;;
```

Slide 14

1. Split-Brain: Dienst auf zwei Rechnern aktiv
2. **Resultat:** wohldefinierter `UP`-Zustand erzwungen

Steuerung der `failsh`

Ein Tcl-Script steuert die Aktionen der `failsh`:

1. Parametrierung der Dienste (Namen, Masters, Slaves, Passwörter)
2. Parametrierung der Hosts (Adressen, Ports, Timeouts)

Slide 15

3. Definition des Traphost und der Trap OID
4. Tcl Prozeduren für Start, Stop und Reclaim

Tcl als momentan verwendete Scriptsprache scheint nicht mehr so gut gepflegt zu sein: 8.3.2 kompiliert nicht auf Solaris 2.6. Daher muss für zukünftige Versionen eine Alternative ins Auge gefasst werden. Als besonders attraktiv präsentiert sich Guile, the GNU ubiquitous intelligent language for extensions, einer einbettbaren Scheme Implementation.

Dienst-Defintion

Schlüsselwort: `failsvc`

Slide 16

```
failsvc test1 create
failsvc test1 interval 5
failsvc test1 start startservice1
failsvc test1 stop stopservice1
failsvc test1 reclaim reclaimservice1
failsvc test1 masters lukretia
```

1. Jeder Dienst muss zuerst kreiert werden, `failsh` legt in diesem Moment eine default-Datenstruktur für den Dienst an. Die späteren Befehle scheitern, wenn die Struktur nicht vorhanden ist.
2. Interval: Zeitabstand, in dem der faild über eventuelle Zustandsänderungen befragt wird.
3. `start`/`stop`/`reclaim` Prozeduren für Start, Stop und Reklamieren des betreffenden Dienstes.
4. Liste der Masters und Slaves

Slide 17

Host-Definition

Schlüsselwort: `failhost`

```
failhost lukretia port 4712
failhost lukretia protocol tcp
failhost lukretia community genesis
```

Wichtig: Definition für `localhost` mit read-write community ist obligatorisch

1. Default-Protokoll ist TCP
2. Default-Port für Zustandsabfragen ist 1291
3. Wenn die Community nicht gesetzt ist, wird `public` verwendet.
4. **Wichtig:** für `localhost` muss eine Definition vorhanden sein mit einem Community String, der Veränderungen erlaubt, die `failsh` kann andernfalls den Zustand im `faild` nicht aufdatieren.

Trap Konfiguration

`failsh` kann bei jeder Zustandsaenderung einen SNMP Trap versenden

Slide 18

```
failtrap port 162
failtrap host netwatcher.doma.in
failtrap community public
failtrap enterprise 1234
failtrap prefix 1.1.1.1
```

Sobald ein Traphost definiert ist, wird `failsh` Zustandsänderungen mit Traps rapportieren.

Mit den Parametern `enterprise` und `prefix` kann die Objektid des ausgesendeten Trap modifiziert werden, zum Beispiel um sie in einem Netzwerk-Management-System besser zu integrieren. Meist ist dies jedoch nicht notwendig, da `failover` eine eigene SNMP Enterprise number zugeteilt erhalten hat.

Als Community wird public genommen, wenn nichts anderes angegeben wird. Dies ist nur notwendig, wenn der Traphost die Traps bei falscher Community abweist.

Demonstration: Geplanter Failover

Slide 19

1. Dienst auf dem Master herunterfahren:

   ```
   /etc/init.d/152.96.232.12 fail
   ```

   ⇒ Master ist jetzt im `FAIL` Zustand (`failsh` kaltgestellt)

2. Slave erkennt Master im `FAIL` Zustand ⇒ führt `UP`-Aktion aus ⇒ `UP`

Resultate:

1. Dienst wird vom Slave angeboten: Backup-Betrieb
2. Wegen Zustand `FAIL` wird Master nicht mehr gestartet

Demonstration: Geplanter Wiederanlauf

Slide 20

1. Dienst wieder startbar machen

   ```
   /etc/init.d/152.96.232.12 start
   ```

   oder

   ```
   /etc/init.d/152.96.232.12 recover
   ```

   ⇒ Master geht auf `RECOVER`

2. Slave erkennt, dass Master wieder verfügbar ist und geht auf `RECOVER`
3. Master sieht, dass Slave abgegeben hat und übernimmt (`UP`)

Resultat: Normalzustand

Demonstration: Stromausfall

1. Simulation: Netzwerkkabel ausziehen
   (a) Slave setzt den Master zunächst in den `INDOUBT`-Zustand, bis Klarheit über Versagen des Masters herrscht
   (b) Sobald der Slave den Master als `FAIL` beurteilt, startet er den Dienst (`UP`).

Slide 21

Resultat: Backup-Betrieb

Demonstration: Strompanne behoben

1. Stromzufuhr wieder OK: Dienst auf dem Master auf `RECOVER` setzen (entspricht der Situation gleich nach dem Booten)
2. Netzwerkkabel einstecken
   (a) Slave erkennt `RECOVER`-Zustand des Master $\Rightarrow$ geht selbst in `RECOVER`
   (b) Master erkennt am `RECOVER`-ZUstand des Slave, dass der Slave den Dienst abgegeben hat $\Rightarrow$ `UP`

Slide 22

Resultat: Normalbetrieb

Demonstration: Split Brain

Slide 23

1. Ausfall des Netzwerkes simulieren: Netzwerkkabel am Master ausziehen
2. Beide Systeme bieten den Dienst an (`UP`)
3. Wiederherstellen des Netzes simulieren: Netzwerkkabel am Master einstecken
4. Reclaim-Szenario
   (a) Slave erkennt Master im `UP` Zustand $\Rightarrow$ `RECOVER`
   (b) Master erkennt Slave im `UP` Zustand $\Rightarrow$ `RECLAIM`
   (c) Master wartet, bis Slave im `RECOVER` ist $\Rightarrow$ Master führt Reclaim-Aktion aus $\Rightarrow$ `UP`
5. Vorteil gegenüber `RECOVER-UP`-Szenario: Sessions auf Master gehen nicht verloren.

Fallbeispiel: DNS

Slide 24

1. Minimale Verfügbarkeitshilfe in den Resolver eingebaut: mehrere DNS Server können spezifiziert werden.
2. Einige PC-Implementationen von TCP/IP sprechen nur den ersten Eintrag an
3. Wechsel vom ersten auf den zweiten Eintrag erfolgt pro Request (nicht pro Prozess): erst nach 12 Sekunden $\Rightarrow$ Anwendungen werden unbrauchbar langsam
4. Anwendungen mit engen Zeitverhältnissen stürzen ab.

$\Rightarrow$ Der Failover für DNS muss für den Resolver transparent gemacht werden.

DNS Eigenheiten

Slide 25

1. Server bindet auf alle IP Adressen, die im System gefunden werden.
2. DNS Antworten sind nur möglich für Anfragen, die an eine der Adressen gerichtet sind
3. Zur Laufzeit können keine neuen Adressen hinzugefügt werden (auch nicht mit `kill -HUP ‘cat /var/named/named.pid‘`) $\Rightarrow$ Neustart erforderlich

DNS Analyse

Slide 26

1. Netzwerkcharakteristik
   (a) Feste Portnummer 53
   (b) UDP Ports gebunden auf allen Adressen des Systems
   (c) TCP-Port für Zonentransfer
2. Verhalten aus Client-Sicht
   (a) DNS Anfragen (UDP) sind sehr langsam, bis Failover abgeschlossen
   (b) Zonentransfers brechen zusammen
3. Serverseitiger Zustand
   (a) kein clientspezifischer Zustand
   (b) Cache: Verlust spielt keine Rolle

(c) Zonendaten: mit Zonentransfer replizierbar, Backupfile vorhanden

(d) Primary und Secondary DNS sind vom Client nicht unterscheidbar

(e) Ist der Master auch Primary, werden die Daten im Slave bei längerem Ausfall des Masters ungültig (expire)

Slide 27

4. Failover Randbedingungen
   (a) Restart des Servers nach Aktivierung der IP-Adresse auf dem Slave
   (b) Umwandlung des Secondary in einen Primary bei längerem Ausfall des Masters
5. Wiederanlauf: keine besonderen Schwierigkeiten

6. Alternativen
   (a) Service-Adresse auf `lo0:1` aktivieren $\Rightarrow$ kein Serverrestart notwendig, nur Umkonfiguration der Adressen.
   (b) Service-Adresse auf beiden Hosts ständig aktiv, Server senden periodisch (60sec) einen gratuituous ARP
      i. Failover-Zeit ~ 30s
      ii. Zonentransfer praktisch unmöglich $\Rightarrow$ beide Server müssen Secondaries sein
      iii. Vorteil: keine Kommunikation zwischen den Server notwendig
      iv. Nachteil: kein TCP basierter Dienst auf der gleichen IP-Adresse möglich

Slide 28

Dass der DNS-Server auf allen Adressen binden muss, ist darin begründet, dass es keine andere Möglichkeit gibt, eine DNS-Antwort von einer bestimmten Adresse abzusenden. Ohne diese bereits gebundenen Sockets wird der abgehenden Antwort diejenige Adresse als Source-Adresse zugewiesen, die zum Interface gehört, für welches sich die Routingtabelle entschieden hat. Diese kann verschieden sein von der Adresse, auf der das Paket empfangen wurde, was dazu führen würde, dass sie vom Klienten nicht akkzeptiert würde.

Genau genommen gibt es natürlich schon so etwas wie Zustand: wenn ein Client einen Namen auflöst, der noch nicht im Cache drin steht, erhält er eine Antwort, die als autoritativ deklariert ist. Bei der zweiten Anfrage wird die Anfrage aus dem Cache heraus beantwortet, also nicht autoritativ. Wenn aber dazwischen ein Failover stattgefunden hat, wird die zweite Anfrage wieder von einem DNS Server beantwortet, der noch nichts im Cache hat, also wieder autoritativ.

Fallbeispiel: Mail-Gateway

1. Verfügbarkeit im MX-Mechanismus des DNS
2. Nicht MX-befähigte Mailer sind so gut wie ausgestorben
3. Wird der Gateway als Smart Host genutzt, spielt der MX-Mechanismus (oft) nicht (Lotus Notes)

Slide 29

4. Keine schnelle Reaktionszeit gefragt: Mail delivery ist ohnehin nur best effort.

⇒ Failover technisch nicht notwendig:

1. Die normalen Mail-Mechanismen können Ausfälle von mehreren Tag abfangen
2. Ausfälle im Internet sind mindestens so wahrscheinlich wie ein Ausfall des Gateway

Mit Smart Host meinen wir einen Rechner, der alle Mails aus dem Intranet entgegennimmt, und das Routing in andere Teile des Unternehmens (zum Beispiel auf der Basis eines LDAP-Verzeichnisses oder einer statischen Tabelle) oder ins Internet (auf der Basis der MX Records) vornimmt. Von den Clients soll dabei so wenig wie moeglich verlangt werden müssen. So kann man zum Beispiel bei einem Perl-Script welches die `Net::SMTP`-Klasse verwendet nicht davon ausgehen, dass MX-Records ausgewertet werden.

Analyse Mail-Gateway

1. Netzwerkcharakteristik
   (a) SMTP-Dienst auf Port 25, nicht auf IP-Adresse gebunden
   (b) Server verwendet AUTH-Protokoll um Identität des Klienten festzustellen

Slide 30

   (c) DNS als Basisdienst (meist) notwendig
2. Verhalten aus Client-Sicht
   (a) SMTP Dialog reagiert nicht mehr, und bricht zusammen, sobald Failover abgeschlossen ist
3. Serverseitiger Zustand
   (a) Meldungen werden auf dem Server gespeichert, sobald der SMTP Dialog erfolgreich abgeschlossen wird

   (b) Race Condition: Tod des Servers in dem Moment, wo er den Empfang dem Klienten bestätigen sollte. Die Message wird unter Umständen zweimal zugestellt
4. Failover Randbedingungen: keine besonderen Anforderungen
5. Wiederanlauf: keine besonderen Schwierigkeiten

Slide 31

6. Alternativen
   (a) Transport-Infrastruktur nach dem Hol-Prinzip (UUCP) statt dem Bring-Prinzip von SMTP
   (b) Nachteil: UUCP Clients praktisch nur in der Unix-Welt
7. Erweiterungen
   (a) Zwei Gateways können sich unterschiedlich verhalten: der Master verwendet SMTP über das Internet, der

Slave UUCP über einen ISDN Terminal-Adapter. Damit können Link-Ausfälle für Mail abgefangen werden.

(b) Beide Gateways verwenden UUCP, der Master über TCP, der Slave über Modem. Damit erreicht man Unabhängigkeit vom Transportmedium

Slide 32

Fallbeispiel: Web-Server

1. HTTP Protokoll zustandslos, modulo Keepalive
2. Keine HA-Vorkehrungen im Protokoll
3. Dienstauswahl über IP-Adresse (IP based virtual host) oder Hostname (name based virtual host)

Slide 33

4. HTTPS immer IP based
5. Redirect genügt nicht: Redirector muss hochverfügbar sein
6. IP based: failover individuell pro virtuellem Server möglich

Analyse Web-Server

1. Netzwerkcharakteristik
   (a) TCP Verbindung, normalerweise gebunden an Port 80, jeder andere Port ebenfalls möglich
   (b) Serverauswahl erfolgt über IP-Adresse oder HTTP Header

Slide 34

   (c) Server kann auf Adresse gebunden sein (notwendig, falls mehrere Serverprozesse auf dem gleichen Host und der gleichen Maschine laufen) $\Rightarrow$ Server kann erst im Failover-Fall gestartet werden
2. Verhalten aus Client-Sicht
   (a) Einzelne Requests schlagen möglicherweise fehl (timeout oder reset), können aber mit reload neu gestartet werden
   (b) Bei Webanwendungen muss unter Umständen von vorne begonnen werden
   (c) Applets: Diskrepanz zwischen clientseitigem und serverseitigem Zustand möglich
   (d) Authentisierung mit Basic Authentication funktioniert weiterhin

Slide 35

   (e) Authentisierung mit Client Zeritifikat muss neu gemacht werden (Session Key geht verloren)
3. Serverseitiger Zustand
   (a) statische Seiten, können durch Replikation synchron gehalten werden
   (b) dynamische Seiten, abhängig von globalen Daten (Zusammensetzen von Seiten aus Templates) oder

klientspezifischen Daten (Datenbankzugriffe)

(c) Anwendungszustand bei Web-Anwendungen (Formularinhalte, temporäre Files, Datenbankeinträge, Cookies)

(d) SSL Session Key (geht verloren)

4. Failover Randbedingungen

Slide 36

(a) Server darf nicht an spezifische Adresse gebunden sein, sonst ist ein Restart notwendig

(b) Vorkehrungen für die Replikation des serverseitigen Zustand notwendig: Datenbankcluster, LDAP-Replikation, shared Filesystem ...

5. Wiederanlauf

(a) Zustandsdaten vom Slave auf den Master bringen:

Datenbankrecovery, LDAP Resynchronisation (nur bei Multi-Master-Replication), Files kopieren

(b) Konsistenz der Zustandsdaten auf dem Master herstellen (Recovery, symbolische Links, Sequenznummern in Cookies etc)

(c) Master starten

Slide 37

6. Alternativen

(a) Mit DNS ein IP-Adresse mit extrem kurzer Time to Live publizieren

(b) Im Failover-Fall DNS-Eintrag modifizieren

(c) Problem: einige Clients halten sich nicht an die TTL

(d) Update der Secondaries möglicherweise nicht kontrollierbar

Fallstudie: File- oder Mail-Server

Slide 38

1. Verschiedene Zugriffsprotokolle: NFS, SMB, FTP, IMAP, POP, RPCs
2. Einzigartiger und sehr grosser Datenbestand: Benutzerfiles, Benutzermailboxen
3. Filesharing: extensive Verteilung von Zustand auf Client und Server. Benutzer öffnet File zur Bearbeitung, schreibt möglicherweise ein Änderungslog, schliesst File (Mailbox analog)
4. Benutzer erwartet, dass Updates sofort wieder zum Lesen verfügbar sind.

Filserver: NFS

Slide 39

1. NFS v2 funktioniert zustandslos
2. Identifikation von Files: Name $\rightarrow$ Status query $\rightarrow$ NFS File Handle
3. File Handle hängt nur vom File (device, inode) ab, nicht vom Namen (Linux user space NFS macht das falsch!)
4. File Handles müssen opak behandelt werden, für den Client ist der Handle einfach ein Bitblock
5. Locks können Clients wieder erfragt werden

Literatur: Brent Callaghan, NFS Illustrated, Addison Wesley Professional Computing Series, 2000

Fileserver: SMB

1. TCP Verbindungen auf Port 139
2. Eintrag in Netbios erfolgt normalerweise mit Hauptadresse des Servers
3. Verbindung wird vom Client sofort wiederhergestellt

Slide 40

4. Identifikation des Files nur über den Namen
5. Extensiver Gebrauch von Locking, bei Samba gespeichert in Shared memory, also nicht replizierbar
6. Windows-Client speichert Passwort, Reauthentisierung ohne Benutzerinteraktion möglich

Fileserver: FTP

1. Mehrer TCP-Verbindungen, jedoch gesteuert über eine einzige Kontrollverbindung auf TCP Port 21
2. Nur ganze Files werden transportiert
3. Replikation praktikabel

Slide 41

4. Verzögerte Replikation eventuell tolerierbar
5. Files werden über Namen identifiziert
6. Kein Zustand ausserhalb der eigentlichen Datenfiles

Mailserver

Slide 42

1. Verschiedenartige Quellen von Updates: MUA(s), MDA
2. Zugriffsprotokoll unterstützt automatischen Reconnect
3. Häufige "kleine" Updates
4. Sehr viele Files (Cyrus IMAP), überall verstreute Files (UW IMAP, popper), oder undurchsichtige Datenbank (Exchange)
5. Praktikabel nur mit einem Cluster-Filesystem oder einer geclusterten Datenbank

Analyse File-/Mail-Server

Slide 43

1. Netzwerkcharakteristik (siehe oben)
2. Verhalten aus Client-Sicht (siehe oben)
3. Serverseitiger Zustand
   (a) grosse Datenmengen
   (b) häufige Updates
4. Failover Randbedingungen
   (a) Failover für Filesystem oder Datenbank notwendig
   (b) Clusterfilesystem (Zukunft)
   (c) NFS Server Backend (warum nicht alles dort?)
5. Wiederanlauf
   (a) Slave anhalten

(b) Datenbestand auf Master restoren,
evtl. Datenbankrecovery

(c) Master starten

6. Alternativen

(a) Benutzer auf mehrere kleine Mailserver verteilen, so dass bei einem Ausfall nur ein Teil betroffen ist.

Slide 44

Murphy: es ist immer der wichtigste Teil, der von einem Ausfall betroffen wird

Fallbeispiel: Datenbank-Server

1. TCP Verbindungen auf festem Service-Port
2. "Hochfrequente Mikroupdates"
3. Redo-Logs und Snapshots können regelmässig repliziert werden

Slide 45

4. Replikation der Updates meist nicht instantan oder dann teuer
5. Spezielle Cluster-Features in kommerziellen Produkten verfügbar
6. MySQL: keine Transaktionen, RI nur applikatorisch gewährleistet
7. Wiederanlauf oft sehr umständlich

Als Illustration ein Beispiel eines Web-Servers mit Datenbank-Backend. In der Datenbank werden einerseits statische Daten (Tarife) gespeichert, andererseits der Zustand der Anwendung: Name und Adresse des Benutzers, was will er eigentlich, was ist ihm schon geboten worden. Wenn der Master ausfällt, arbeitet die Anwendung in der Slave-Datenbank weiter. Diese ist mit Replikationsmechanismen (im konkreten Fall mit DB2 Replikation) aufdatiert gehalten worden. Beim Wiederanlauf muessen eventuell im Slave erzeugte Kunden-Einträge (zum Beispiel Offerten) nachgetragen werden. Jeder Kunde ist mit einer Sequenznummer ausgestattet, die erlaubt, alle für ihn durchgeführten Aktionen zuzuordnen. Diese werden aber von der **Anwendung** generiert, der Master darf also erst wieder anlaufen, wenn alle Einträge vom Slave auf den Master transportiert worden sind, und die Sequenznummergenerierung auf einen sicheren Stand gebracht worden ist. Recovery ist also bedeutend komplizierter als Failover.

Slide 46

Fallbeispiel: Firewall

1. Verschiedene Arbeitsweisen von Firewalls: Paketfilter, stateful packet filter, Proxies, transparente Proxies
2. Paketfilter haben keinen Zustand: Failover jederzeit möglich
3. stateful packet filter haben geringe Zustandsmengen, Echtzeitreplikation denkbar (Firewall-1 State Synchronization)
4. VPN-Gateways: meist kryptographisch geschützte Sequenznummern zusätzlich zu den Session Keys

Analyse Firewall: Linux `ipchains`

1. Netzwerkcharakteristik
   (a) Paketfilter arbeitet zustandslos
   (b) Masquerading verwendet eine Verbindungs-Tabelle
2. Verhalten aus Client-Sicht
   (a) Alle Verbindungen werden langsam, bis der Failover abgeschlossen ist.
   (b) Verbindungen, für die Masquerading notwendig ist, brechen zusammen

Slide 47

3. Serverseitiger Zustand
   (a) Routing-Tabelle statisch oder von Routingprotokoll aufgebaut
   (b) Session-Tabelle des Masquerading-Moduls
   (c) FreeS/WAN Schlüssel
4. Failover Rahmenbedingungen
   (a) Kurze Schlüssellebensdauer festlegen, damit Renegotiation schnell stattfinden kann
5. Alternativen
   (a) Routing-Protokoll einsetzen (Sicherheitsfeatures wie Authentisierung oder Verschlüsselung ausnützen)
   (b) Beide Firewalls aktiv halten, Wahl des Weges dem Zufall und dem ARP-Protokoll überlassen

Slide 48

6. Erweiterungen
   (a) Linux Virtual Server: Hochverfügbare Firewalls als Load Balancer für einen load sharing cluster einsetzen
   (b) Firewall kann auch Zustand von Anwendungsservern

in DMZ ermitteln, und mit Hilfe von statischem ARP den Failover für Anwendungsserver durchführen

Slide 49

Fallstudie: Applikationsserver

Beispiele: Zope und Apache Jserv

Slide 50

1. Verschiedene Zugriffsmechanismen: standalone Web-Server oder Server für FastCGI (Zope) oder eigenes Protokoll (Apache Jserv)
2. Integrierte monolithische Objektdatenbank (Zope) $\Rightarrow$ Zustandsreplikation muss applikatorisch vollzogen werden
3. Nicht wiederherstellbarer Zustand in der Servlet-Engine $\Rightarrow$ HA Support muss in die Servlets eingebaut werden (zum Beispiel Serialisierung in eine Datenbank und Reaktivierung auf dem Backupsystem)

`failover` Sicherheit: aktuelle Version

1. Zustandsänderungen authentisiert (SNMP Community)
2. `failsh` liest nur vom Debug socket, auf die anderen Sockets wird nur geschrieben
3. `faild` muss nicht als `root` laufen

Slide 51

4. `failsh` muss nicht als `root` laufen, aber die Aktionen (start/stop) müssen durch den Runtime-User ausführbar sein

`failover` Sicherheit: past and future

1. Versionen vor 0.5.0:
   (a) Kein Authentisierung
   (b) Zustandsänderungen nur von 127.0.0.1 akzeptiert
   (c) Debugänderungen nur von 127.0.0.1

Slide 52

2. Zukünftige Versionen
   (a) Authentisierung der Debug- und Monitor-Verbindungen
   (b) TLSv1 für TCP-Verbindungen

Mehrfache Heartbeats

Slide 53

1. Ausfall eines einzelnen Netzwerklinks soll keinen Failover hervorrufen.
2. Slave wird aktiv, wenn alle Masters tot sind
3. `failover` stört es nicht, wenn sich hinter mehreren Namen/IP-Adressen der gleich `faild` verbirgt
4. Für jedes Interface einen Master definieren, im Slave alle Interfaces als Masters eintragen. Analog im Slave
5. Nur der Ausfall des Master-**Rechners** oder **aller** seiner Netzwerk-Interfaces (was aus Client-Sicht dasselbe ist) führt zum Failover