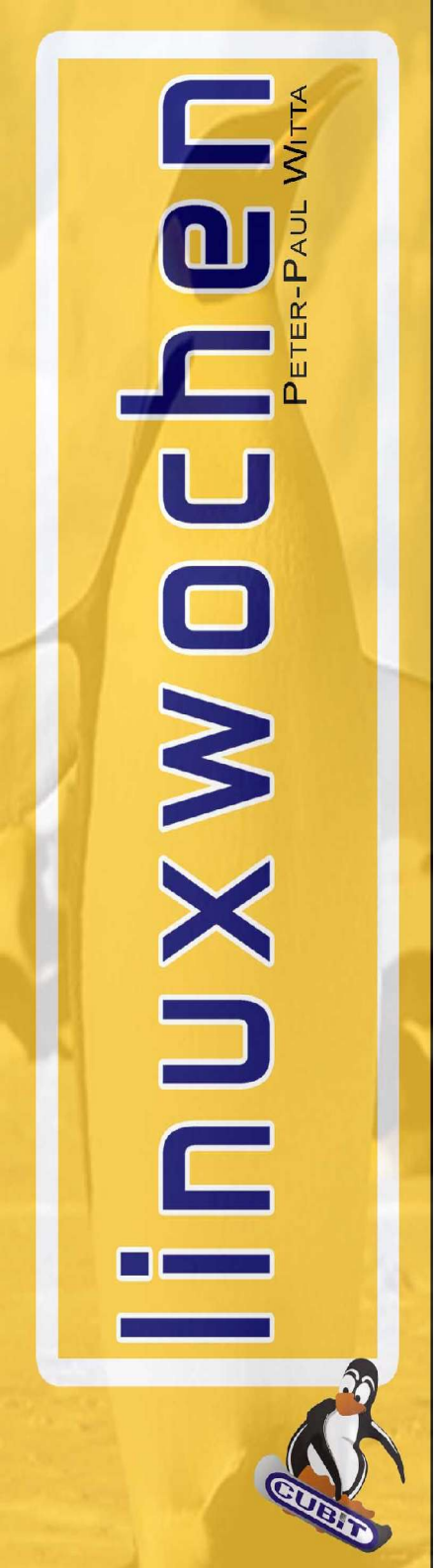


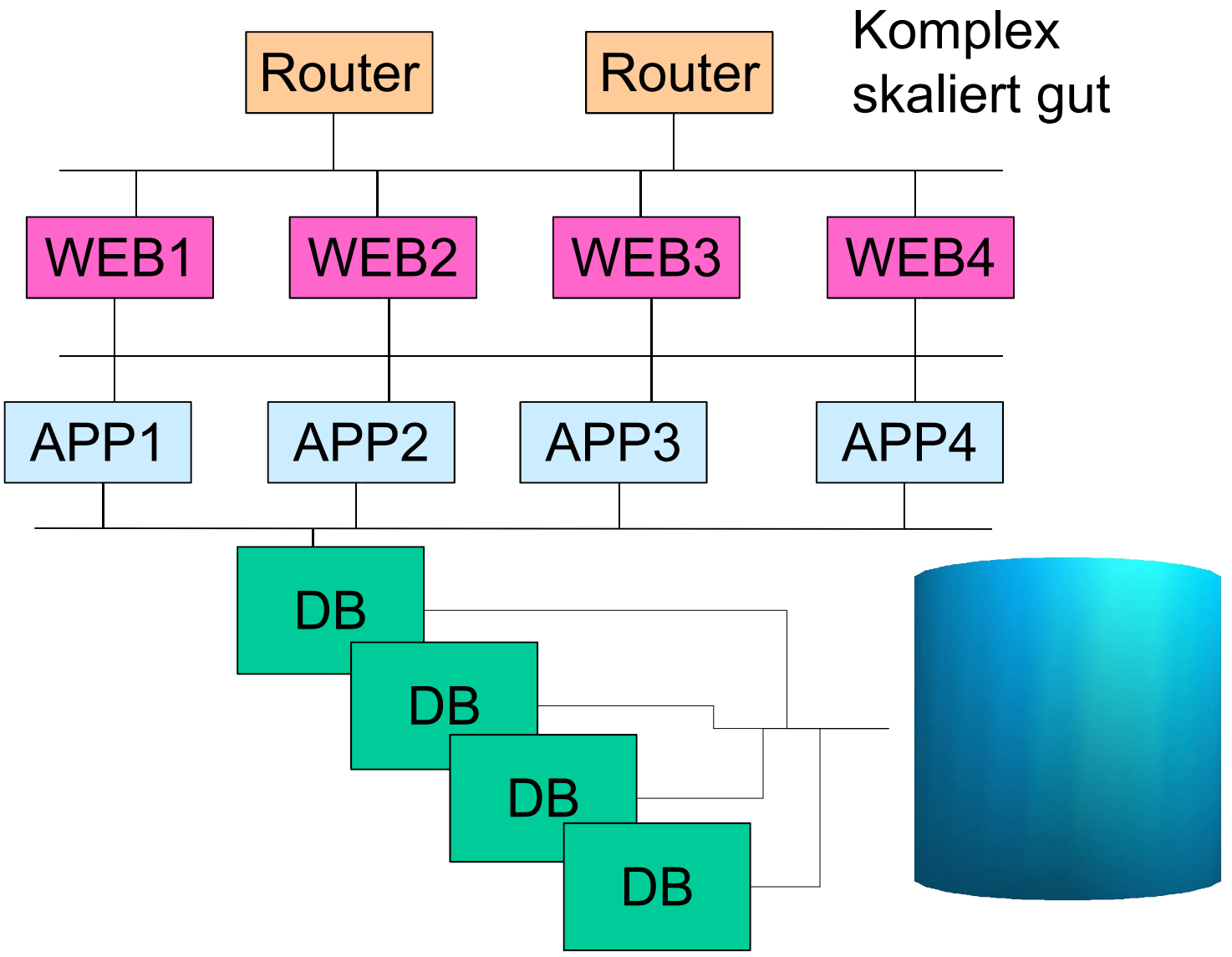
Load Balancing Web Service Architecture

Ing. Peter-Paul Witta
paul.witta@CUBiT.at

Clustering

- mehrere kleine Rechner leisten gemeinsam Grosses
- günstige dual intel/AMD Server
- load sharing
- High Availability
- combined architecture
- centralized management

complex Architectures



Architecture (lg concept)

- load group=(serivce+Ressourcen)
- = (service+network+disk)
- enthält alle Ressourcen für Service
- unteilbar, kann nur als Ganzes einem Server zugeordnet werden
- mehrere lg pro Server möglich
- mehrere Services pro lg möglich
- mehrere lg pro Service möglich

Load Balancing

- incoming Connections Server zuweisen
  - RR-DNS
  - lvs
- Anwendungsbezogen
  - backhand
  - Front-End Reverse Proxy
- Scheduling
  - load-bezogen
  - Anwendungsbezogen
- Datenintegrität
- Synchronisierung
- Auslagern von Speicher

Round Robin DNS (RR-DNS)

- Round-Robin DNS
- mehrere A Records pro FQDN
- Client macht Lookup
- Nameserver retourniert RR-Scheduled IP-Addressen
- CNAME möglich
- nur gleiche Verteilung
- Caching on Lookups
- ungleiche Belastung pro Client
- Proxies, Poweruser
- immer alle Adressen aktiv halten

Frontend Reverse Proxy

- HTTP Proxy von vorn nach hinten
- mit und ohne Cache
- Steuerbar, aber wie?
- Verteilung über Frontnode
- Frontnode muss geclustert werden
- Frontnode limitiert in Performance
- Antworten via Frontend
- teilweise Transparent für Anwendung
- Caching kann dynamisches Generieren von Seiten “abnehmen”, z.B. nur alle 60 Sek. neu anfordern

LVS
IP Direktoren
(Alteon, CISCO)

- IP Router
- Verteilung über Frontnode
- Frontnode muss geclustert werden
- Frontnode limitiert in Performance
- Antworten entweder direkt oder via Frontend
- Direkt: unsaubere IP-Basteleien
- Indirekt: Performanceproblem
- Transparent für Anwendung
- Problem für Session-Daten
- User kann migriert werden, aber wo bleibt seine Session?

mod_backhand

- Apache Modul
- gezielt steuerbar im Apache Configuration Kontext
- Pro URL, damit Pro Script (PHP, Perl,...) direkt ansprechbar
- Load Balancing mit Proxy, d.h.der User bleibt bei “seinem” Server
- Request Proxy
- nur für dynamische Seiten sinnvoll
- kein dedizierter Frontnode notwendig

mod_backhand

- Status-Tool integriert

Entry	Hostname	Age	Address	Total Mem	Avail Mem	# ready servers/ # total servers	~ms/req [#req]	Arriba	# CPUs	Load/HWM	CPU Idle
0	cube5b.cubit.at	0	80.78.231.74:80	1008 MB	817 MB	0/0	89 [377]	1760812	2	0.540/16	0.000
1	cube5c.cubit.at	0	80.78.231.66:80	1008 MB	738 MB	0/0	0 [0]	1760812	2	0.530/16	0.952
2	cube6c.cubit.at	0	80.78.231.67:80	1008 MB	865 MB	0/0	0 [0]	1760812	2	0.230/16	0.986
3	www-lgb.cubit.at	0	80.78.231.75:80	1008 MB	801 MB	8/15	9 [959]	1760526	2	0.360/16	0.000

High Availability

- Ausfallsicherheit durch Redundanz
- duale Server
- jeder “backt” den Anderen
- pro Dienst nur ein Server aktiv
- hohe Leerkosten durch Mehr-Hardware
- Bei Ausfall Migration der Dienste auf Ersatzsystem
- cold standby
- hot standby
- active/active
- Umschaltzeiten
- Session-Verlust(?)

Performance-Analyse

- 1. CGI (ohne FastCGI) => nicht genügend
- 2. PHP=Perl (für diese Betrachtung)
- Execution Times Static/PHP = 1/100
- Execution Times PHP/SQL=1/1000
- Werbserver + Anwendung meist nur DB-Frontend
- Richtige Aufgabe: Scale the DB

Scale the DB

- Analyisieren Access Pattern
- viele Queries/Page Impression
- Entlasten:
  - static Pagecache (keine Zugriffe)
  - Caching in Middleware
  - MySQL: Query Cache
- Skalieren: pro Webserver mind. 1 DB Server
- R/W=n/1 (n sehr gross)
- viele R/O Replikas
- integrierte MySQL R/O Replikas helfen
- static ist am Besten
- Nachteil: Update Delay

go static!

- static scales best
- Personalisierungsproblem
- “building blocks” vorgenerieren
- Portal-Engines setzen Seiten zusammen
- z.B. Slashdot: fast 100% statisch
- z.B. Lion.CC: die wichtigesten Files statisch

Filesystem: NFS, Rsync

- Shared Filesystem NFS vs. Replication
- Replika schnell, aber nicht synchron
- NFS mit Statcache
  - rsize, wsize
  - actimeo=n (Attribute Cache Directory und File)
  - nocto (Attribute Cache bei File creation)
- Performance-kritische Daten replizieren
- rsync is your friend

Building Blocks in complex environments

- Filesystemproblem
- Speicher teuer
- NFS-System: NetApp Filer
- Load Balancer: Alteon, Cisco, LVS & Co
- oder in Anwendung: mod_backhand
- RR-DNS
- Auf jeder Ebene Clustern
- Management wichtig
- Replikationsfähige Datenbank: MySQL

Management-Tools

- ex?: schnelles Ausführen von Commands auf fremden Knoten im aktuellen Pfad
- to?: Kopieren von Files auf fremden Knoten im aktuellen Pfad
- sync-machines: rsyncen von Files basierend auf Timestamps
- Sym-Links: zu synchronisierende DIR auf shared-storage linken
- etc weitestgehend synchron halten
  - für grosse Cluster: Macro-Preprocessing
- Nagios zur Fehlersuche

Examples

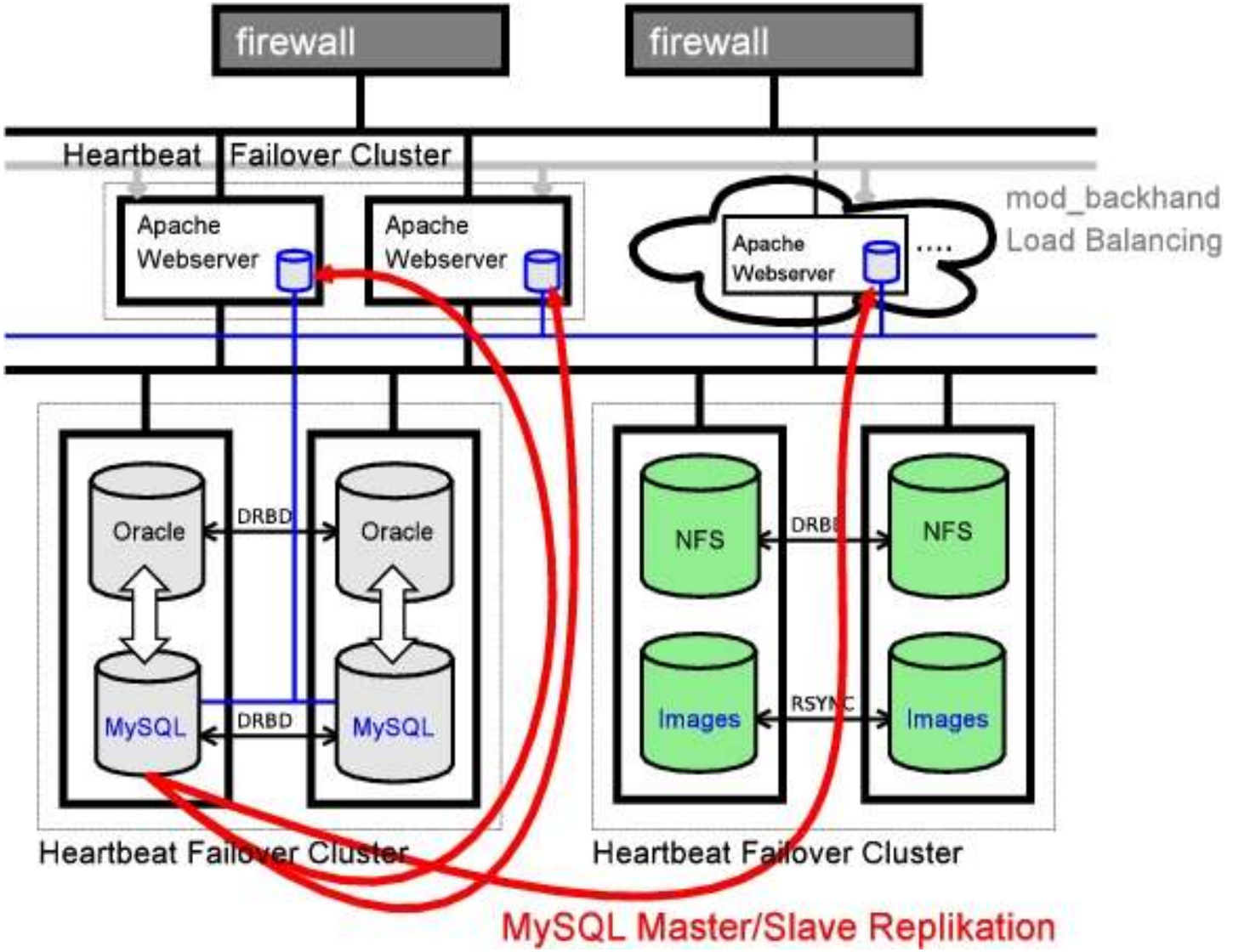
- Hosting Environment CUBiT IT
- www.lion.cc
- 4 Webserver (skaliert linear)
- 2 MySQL Server
- 4 MySQL Slave Replikas
- 2 Oracle Server
- 2 NFS Server

Beispiel Lion.CC

- 2.5 Mio Images (4 Auflösungen)
- dynamisches Caching
- Backend Job lernt und cached alles abgefragte
- Cache für nicht personalisierte Hauptseiten
- Tuning mittels Log-Stats: wichtigste Seiten tunen
- Redirect auf statische Objekte erlaubt Caching
- HTTP-Header erlauben Cachine
  - DB Aggregation auf Webserver: lokal lesen
  - zentral schreiben
  - MySQL+Oracle
    - MySQL für Web-Betrieb
    - Oracle für Businesslogik

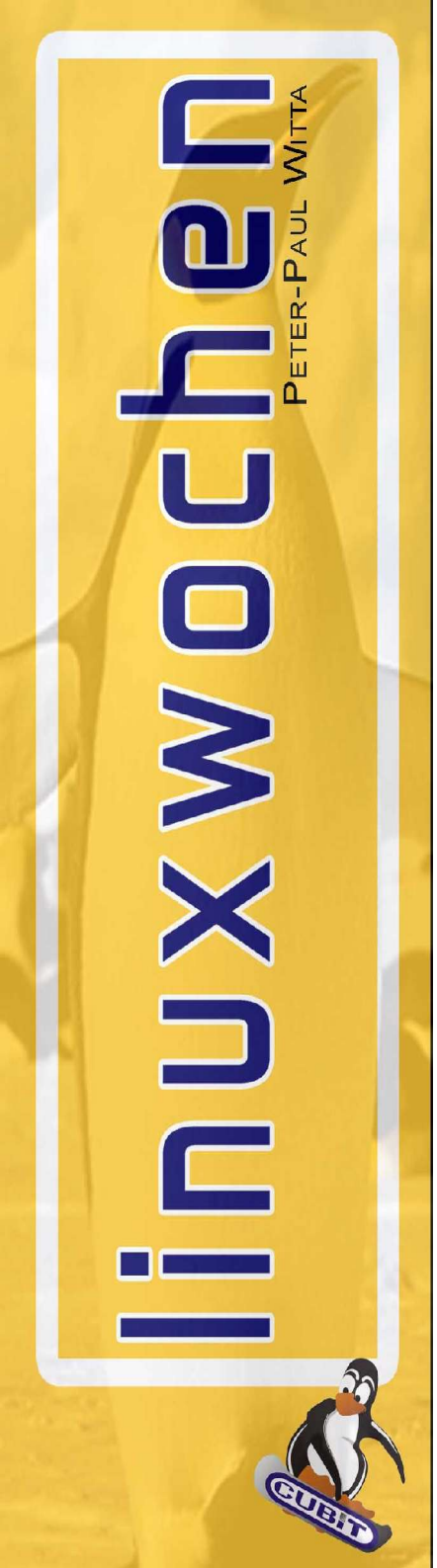
linuxwochen
Peter-Paul Witta

Beispiel Lion.CC



Beispiel Lion.CC

- Clustering mit MySQL bringt schnelle Seektime:
  - Fulltext Query von 180 Sek auf typisch 0.8 Sek
- was braucht Zeit?
  - Banner, Logging, DB-Statistiken
  - Haupt-DB Knoten nur zum Schreiben
  - Replikas zum Lesen
  - Erfahrungen mit 6 Monaten MySQL Replikation:
  - semi-synchron, schnell, performant
  - manchmal nicht stabil



Load Balancing System Architecture

DANKE!

Ing. Peter-Paul Witta
paul.witta@CUBiT.at