Easy Geo-Redundancy with MARS + systemd
Easy Geo-Redundant Handover+Failover: Agenda

- Short background: MARS for end-to-end SLA 99.98%
- Cluster management for long distances
- Using systemd as a cluster manager
- Current Status / Future Plans
**SLA:** 99.98% end-to-end measured from Frankfurt
- Including WAN outages, PHP problems, HumanError™
  
  => MARS geo-redundancy must compensate much better!

- 4 datacenters at 2 continents, pair distance > 50 km
- ~ 9 millions of customer home directories
- ~ 10 billions of inodes + daily incremental backup
- > 4.7 petabytes *allocated* in ~ 3800 xfs instances
  
  LocalStorage LVM ~ 8 PB x 2 for geo-redundancy via MARS
  
  https://github.com/schoebel/mars

- Growth rate ~ 21 % / year
- Solution: Container Football on top of MARS
  
  https://github.com/schoebel/football
Cluster Management for > 50 km

- **Proprietary** e.g. 1&1 cm3 (no GPL)
- **Pacemaker & co typically** don’t work as expected
  - original HeartBeat DSM model: *shared disk*
    cannot *really* handle Split Brain
  - explainable by CAP theorem

- **Using systemd as a Linux cluster manager**
  - already in use almost everywhere e.g. startup of VMs
  - itself somewhat „monolithic“, but extensible via Unit Files
  - path watchers can monitor /mars/resource-$res/* remote updates
    from MARS cluster communication => **generic remote control**

- **MARS dynamic resource creation / deletion**
  marsadm join-resource / leave-resource

- **Solution:** marsadm internal macro processor
  creates / deletes systemd units „on the fly“
systemd Unit Example (Template)

bash> cat \^{mntname}\-@\{res\}.mount
@eval{%let{mntpath}{%subst{%{mntname}}{-}{/}}}  
[Unit]
Description=MARS local mount on /@{mntpath}/@{res}
Requires=mars.service
After=mars.service

ConditionPathIsSymbolicLink=/mars/resource-@{res}/systemd-want
ConditionPathExists=/mars/resource-@{res}/userspace/systemd-want-@{host}
ConditionPathExists=/dev/mars/@{res}
ConditionPathIsDirectory=/@{mntpath}/@{res}

[Mount]
What=/dev/mars/@{res}
Where=/@{mntpath}/@{res}

[Install]
WantedBy=mars.service

Use MARS' metadata symlink updates for remote control
Usage of systemd unit templates

- **Activation of template** *(once after resource creation, for the whole cluster)*
  
  
  ```
  marsadm create-resource $resource /dev/$vg/$resource
  mkfs.xfs /dev/mars/$resource
  marsadm set-systemd-unit $resource $start_unit $stop_unit
  => automatic instantiation via macro processor
  ```

- **Usage at planned handover:**
  
  ```
  marsadm primary $resource (or marsadm primary all)
  ```

  - **Automagically** *(independently for each resource):*
    - Old primary: `systemctl stop $stop_unit`
    - Old primary: MARS goes to secondary mode
    - New primary: MARS becomes primary `/dev/mars/$resource will appear`
    - New primary: `systemctl start $start_unit`

- **Usage at unplanned failover:**
  
  ```
  marsadm disconnect all ; marsadm primary --force all
  ```

  remote control: piggyback on distributed MARS symlinks
BETA feature! not yet in production
- example templates in systemd/ subdir

Currently works **sequentially**
- observation: systemctl is non-reentrant, can deadlock
- `marsadm` uses (breakable) locks for protection

Planned improvements: all resources in parallel to each other
- Needs heavy testing
- Help from the community welcome!
- e.g. contribute new systemd templates for KVM startup, or iSCSI / NFS exports, ...
MARS Current Status

- MARS source under GPL + docs:
  
  - [github.com/schoebel/mars](https://github.com/schoebel/mars)
  - [docu/mars-architecture-guide.pdf](https://docu/mars-architecture-guide.pdf)

- mars0.1 stable productive since 02/2014
- Backbone of the 1&1 Ionos geo-redundancy feature
- up to 14 LXC Containers on 1 Hypervisor
  - Efficiency project using Football:
    - TCO has **halved!**
Faster checksumming (CRC32c | CRC32 | SHA1 | MD5)
Logfile compression (LZO | LZ4 | ZLIB)
Optional network transport compression
  - may help for some very slow networks
  - IO data paths already scaling well

TODO: better metadata scalability needed!
  - single `mars_main` control thread (non-blocking)
  - TODO: more resources per host (max. 24 in prod at 1&1)
  - TODO: more hosts per cluster

TODO: Linux kernel upstream
  - requires a lot of work!
  - tomorrow's presentation at kernel miniconf

TODO: more tooling, more systemd templates, integration into other OpenSource projects, ...

Collaboration sought

=> Opportunities for other OpenSource projects!
Sponsoring (MARS + Football)

- Best for > 1 PiB of enterprise-critical data
  - More Football plugins in future, e.g. for KVM, ...

- Future pool-optimizer will deliver similar functionality than Kubernetes
  - but on stateful storage + containers instead of stateless Docker containers
  - State is in the storage and in the machines, but not in orchestration

- Long-term perspective
  - MARS is largely complementary to DRBD
  - Geo-redundancy with OpenSource components
  - distances > 50km possible, tolerates flaky replication networks
  - Price / performance better than anything else (see mars-architecture-guide.pdf)
  - Architectural reliability better than BigCluster with cheaper hw + network!

- ask me: decades of experience with enterprise-critical data and long-distance replication
Why GEO-Redundancy

- Example: GALILEO incident (DR / CDP did not work)
  - Disaster = earthquake, flood, terrorist attack, power outage, ...

- BSI Paper 12/2018:
  Kriterien für die Standortwahl höchstverfügbärer und georedundanter Rechenzentren
  
  [link]

  in English: Criteria for Locations of Highly Available and Geo-Redundant Datacenters
  
  - Stimulated some controversial discussions, but see commentary
    [link]

- Conclusions: distances > 200 km „recommended“
- Might influence future legislation (EU / international)
- „Critical Infrastructures“ more important!
Synchronous does not generally work over ≈50 km
- like iSCSI over 50 km

Need Asynchronous Replication
- Application specific, e.g. mySQL replication
- Commercial appliances: $$$ €€€
- OpenSource
  - plain DRBD is NOT asynchronous
    - commercial DRBD-Proxy: RAM buffering
  - MARS: truly asynchronous + persistent buffering + transaction logging + MD5 checksums + Anytime Consistency
Network Bottlenecks: MARS

- MARS network throughput
- MARS application throughput
- TCP send buffer way too small
- flaky throughput limit
- Best possible throughput behaviour
- corresponding DRBD inconsistency
**Replication at Block Level vs FS Level**

**Userspace Application Layer**
- Apache, PHP, mySQL, Mail Queues, etc

**Filesystem Layer**
- xfs, ext4, btrfs, zfs, ...
- vs nfs, Ceph, Swift, ...

--- **NO long distances**

**Caching Layer**
- Page Cache, dentry Cache, ...

**Block Layer**
- LVM, DRBD / MARS

+++ **LONG DISTANCES**

**Hardware**
- Hardware-RAID, BBU, ...

--- **Potential Cut Point A**
- for Distributed System
- ~ 25 Operation Types
- ~ 100,000 Ops / s

--- **Potential Cut Point B**
- for Distributed System
- DSM = Distributed Shared Memory
- => Cache Coherence Problem!

--- **Potential Cut Point C**
- for Distributed System
- +++ replication of VMs for free!
The CAP Theorem states that in a distributed system:

- **C** = Strict Consistency
- **A** = Availability
- **P** = Partitioning Tolerance

Any two of these three properties may be achieved, but not all three simultaneously. Violations can occur in:
- Disasters
- Long distances

Pick any two.
Network Bottlenecks (1) DRBD

- network throughput

- DRBD throughput

- (potential) incident → automatic disconnect

- automatic re-connect

- mirror inconsistency ...

- additional throughput needed for re-sync, not possible

- wanted application throughput, not possible

- decreasing throughput limit

- Permanently inconsistent!
Network Bottlenecks (2) MARS

- network throughput
- replication network throughput
- application throughput, recorded in transaction log
- Best possible throughput behaviour at information theoretic limit
- decreasing throughput limit

MARS FROSCON 2015 Presentation by Thomas Schöbel-Theuer
MARS Data Flow Principle

Host A (primary)
/dev/mars/mydata

Transaction Logger
Temporary Memory Buffer

writeback in background
append
/dev/lv/mydata
/mars/resource-mydata/log-00001-hostA

Long-distance transfer

Host B (secondary)

Logfile Replicator
/mars/resource-mydata/log-00001-hostA

Logfile Applicator
/dev/lv/mydata
Use Cases DRBD+proxy vs MARS

**DRBD+proxy**
- Application area:
  - Distances: any
  - Aynchronously
    - Buffering in RAM
  - Unreliable network leads to frequent re-syncs
    - RAM buffer gets lost
    - at cost of actuality
  - Long inconsistencies during re-sync
  - Under pressure: permanent inconsistency possible
  - High memory overhead
  - Difficult scaling to k>2 nodes

**MARS**
- Application area:
  - Distances: any ( >>50 km )
  - Asynchronously
    - near-synchronous modes in preparation
  - Tolerates unreliable network
  - Anytime consistency
    - no re-sync
  - Under pressure: no inconsistency
    - possibly at cost of actuality
  - Needs >= 100GB in /mars/ for transaction logfiles
    - dedicated spindle(s) recommended
    - RAID with BBU recommended
  - Easy scaling to k>2 nodes
DRAM+proxy Architectural Challenge

DRBD Host A (primary)  Proxy A'  DRBD Host B (secondary)

Proxy B' (essentially unused)

bitmap A

huge RAM buffer

bitmap B

A != A' possible

data queue path (several GB buffered)

completion path (commit messages)

sector #8

#8

#8

#8

#8

#8

same sector #8 occurs $n$ times in queue

$n$ times

=> need $\log(n)$ bits for counter

=> but DRBD bitmap has only 1 bit/sector

=> workarounds exist, but complicated

(e.g. additional dynamic memory)
Badly Scaling Architecture: **Big Cluster**

**Internet**  
\[O(n \times k)\]

**Internal Storage (or FS) Network**  
\[O(n^2)\] REALTIME Access like cross-bar

**Frontends**
- Frontend 1
- Frontend 2
- Frontend 3
- Frontend 4
- Frontend 5
- Frontend 6
- Frontend 999

**Storages**
- Storage 1
- Storage 2
- Storage 3
- Storage 4
- Storage 5
- Storage 6
- Storage 999
Well-Scaling Architecture: **Sharding**

- **Internet** $O(n \times k)$  

**Smaller Replication Network for Batch Migration**  $O(n)$  

++ local scalability: spare RAID slots, ...  

+++ traffic shaping possible  

=> method **really** scales to petabytes  

2x for geo-redundancy
2 Node failure => ALL their disks are unreachable

DRBD or MARS simple pairs

=> no customer-visible incident

Low probability for hitting the same pair, even then: only 1 shard affected => low total downtime

Big Storage Cluster e.g. Ceph, Swift, ...

k=2 replicas not enough => INCIDENT because objects are randomly distributed across whole cluster

same n

Higher probability for hitting any 2 nodes, then O(n) clients affected => much higher total downtime

need k >= 3 replicas here

O(n²) network
<table>
<thead>
<tr>
<th>Cost (1) non-georedundant, n&gt;100 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Big Cluster:</strong> Typically ≈RAID-10 with k=3 replicas for failure compensation</td>
</tr>
<tr>
<td><strong>Disks:</strong> &gt; 300%</td>
</tr>
<tr>
<td><strong>Additional CPU and RAM for storage nodes</strong></td>
</tr>
<tr>
<td><strong>Additional power</strong></td>
</tr>
<tr>
<td><strong>Additional HU</strong></td>
</tr>
<tr>
<td>*<strong>Simple Sharding:</strong> Often local RAID-6 sufficient (plus external backup, no further redundancy)</td>
</tr>
<tr>
<td><strong>Disks:</strong> &lt; 120%</td>
</tr>
<tr>
<td><strong>Client == Server no storage network</strong> MARS for LV background migration</td>
</tr>
<tr>
<td><strong>Hardware RAID controllers with BBU cache on 1 card</strong></td>
</tr>
<tr>
<td><strong>Less power, less HU</strong></td>
</tr>
</tbody>
</table>
Big Cluster:
- $2X \approx \text{RAID-10}$ for failure compensation
  ($k=6$ replicas needed, smaller does not work in long-lasting DC failure scenarios)

Disks: > 600%

Additional CPU and RAM for storage nodes

Additional power

Additional HU

Geo-redundant Sharding:
- $2 \times \text{local RAID-6}$
- MARS for long distances or DRBD for room redundancy

Disks: < 240%

Hardware RAID controllers with BBU

Less power

Less HU
### Cost (1+2): Geo-Redundancy Cheaper than Big Cluster

<table>
<thead>
<tr>
<th>Single Big Cluster:</th>
</tr>
</thead>
<tbody>
<tr>
<td>≈RAID-10 with k=3 replicas for failure compensation</td>
</tr>
<tr>
<td>O(n) Clients</td>
</tr>
<tr>
<td>+ 3 • O(n) storage servers</td>
</tr>
<tr>
<td>+ O(n²) storage network</td>
</tr>
<tr>
<td>Disks: &gt; 300%</td>
</tr>
<tr>
<td>Additional power</td>
</tr>
<tr>
<td>Additional HU</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Geo-redundant sharding:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x local RAID-6</td>
</tr>
<tr>
<td>MARS for long distances or DRBD for room redundancy</td>
</tr>
<tr>
<td>2 • O(n) clients = storage servers</td>
</tr>
<tr>
<td>+ O(n) replication network</td>
</tr>
<tr>
<td>Disks: &lt; 240%</td>
</tr>
<tr>
<td>Less total power</td>
</tr>
<tr>
<td>Less total HU</td>
</tr>
<tr>
<td>+++ geo failure scenarios</td>
</tr>
</tbody>
</table>
Costs (3): Geo-Redundancy even Cheaper

Precondition: CPU must not be the bottleneck

Idea: passive LV roles get less CPU

1 datacenter out of 3 may fail

Total Storage: \(x\ 2\)
Total CPU: \(x\ 1.5\)
\(\Rightarrow\ 1.5 \cdot O(n)\)

HOWTO flexible CPU assignment \(\Rightarrow\) next slide
Flexible MARS Sharding + Cluster-on-Demand

any hypervisor works in client and/or server role
and preferably **locally** at the same time
Flexible MARS Background Data Migration

Any # replicas $k=1,2,3,…$ dynamically creatable at any time and anywhere

=> any hypervisor may be source or destination of some LV replicas at the same time
Football Current Status

- GPL with lots of plugins, some generic, some 1&1-specific
  - about 2/3 of code is generic
  - plugins/football-basic.sh uses systemd as cluster manager
  - https://github.com/schoebel/football
  - https://github.com/schoebel/mars

- Multiple operations:
  - migrate $vm $target_cluster
    - low downtime (seconds to few minutes)
  - shrink $vm $target_percent
    - uses local incremental rsync, more downtime
  - expand $vm $target_percent
    - online, no downtime

In production at 1&1 Ionos
- get rid of old hardware (project successfully finished)
- load balancing
- >50 „kicks“ per week
  - limited by hardware deployment speed
  - Proprietary Planner (for HW lifecycle)