Motivation: Scalability of Storage Architectures

Motivation: Reliability Unexpected properties!

HOWTO Container Football = Background Migration of LVs
e.g. for load balancing, HW lifecycle, etc

The Football Automation Project

Current Status / Future Plans

New method for load balancing
Badly Scaling Architecture: **Big Cluster**

Data already partitioned + isolation needed

**Internet**  O(n*k)

**Internal Storage (or FS) Network**

O(n^2) REALTIME Access like cross-bar

- User 1
- User 2
- User 3
- User 4
- User 5
- User 6
- User 7
- User 8
- User 9
- User 10
- User 11
- User 12
- User 13
- User 14
- ...  
- User 999999

- Frontend 1
- Frontend 2
- Frontend 3
- Frontend 4
- Frontend 5
- Frontend 6
- ...  
- Frontend 999

- Storage 1
- Storage 2
- Storage 3
- Storage 4
- Storage 5
- Storage 6
- ...  
- Storage 999

For geo-redundancy x2
Well-Scaling Architecture: **Sharding**

- **Internet**: O(n*k)
- **Smaller Replication Network** for Batch Migration: O(n)
- +++ traffic shaping possible
- ++++ big scale out
- +++ local scalability: spare RAID slots, ...
- => method *really* scales to petabytes
Reliability of Architectures: NODE failures

2 Node failure => ALL their disks are unreachable

DRBD or MARS
simple pairs

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

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

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

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

need k >= 3 replicas here

=> no customer-visible incident

same n

O(n²) network
Architectural Reliability Example

- **Same / comparable dimensioning for BigCluster vs Sharding**
- **Simplified assumptions** (more details in mars-manual.pdf):
  - 1 server has 99.99 % uptime => incident probability $p = 0.0001$
    => 1 hour downtime per 10,000 operation hours $\approx 13$ months $\approx 1$ year
  - Only temporary failures, no dependencies between servers
  - $x$ axis = # application units = #VGs ~ # LVs
  - $k$ = number of replicas => price tag
  - Sectors/Objects are dependent (e.g. classical filesystems on top of LVs)
    - maybe too conservative, but NO GAMBLING! (timely or spatially)
  - BigCluster random replication:
    - all objects uniformly spread to all servers
    - „many“ objects per server => otherwise price tag!
  - Sharding (DRBD or MARS): simple pairs / triples / ...
    - 10,000 servers => always 1 of 10,000 is down in average

MARS Presentation by Thomas Schöbel-Theuer
Look at 1 LV (from many), then
Sharding with pairs / triples / etc has the
BEST POSSIBLE RELIABILITY.

- mathematical proof sketch (induction) at mars-manual.pdf
  motivated by practical experiences with 1&1 Ceph clusters

- BigCluster random replication (same k) is never better.
  - even worse when considering storage network outages, frontend
    node failures, permanent failures / disasters, etc.

- don‘t neglect k, the price tag!
- don‘t burn arbitrary holes into LVs at (fatal) incidents!

Result is contrary to some common belief
Fundamental Law (2)

Workarounds for BigCluster:
- USENIX paper on copysets (2013)
- Buckets, spread factors, partitioning, etc
  - idea: spread objects to only $O(k)$ instead of to $O(n)$ storage nodes
    but worse than Sharding on same $k$
- Erasure encoding? => similar to RAID-60 etc, but on $O(n)$ Bigcluster???

Better formulation of general law:
- Distribute your LV data to as less nodes as possible!
  Spreading to more than necessary worsens reliability
  known as RAID-0 problem
- Replicate $k$ into separate failure domains / over long distances

Smallest BigCluster spread: result is then similar to sharding,
likely needs similar load balancing / data migration over time
Common Belief

- Sharding is inflexible / no load balancing possible???
  - therefore storage networks are a „must“???

- Yes, maybe in the past

- NO LONGER in future => see new Football method
  - VM Football / Container Football / LV Football / ...

Common belief changes only slowly
But fundamental laws of physics / mathematics are stronger
HOWTO Container Football = Background Migration of LVs

HOST A (old) VM is running
- lvdisplay /dev/vg/$mydata
- (meanwhile VM is altering data)
- $vmmanager stop /dev/mars/$mydata
- marsadm leave-resource $mydata
- lvremove /dev/vg/$mydata

HOST B (new) has spare space
- lvcreate -L $size -n $mydata vg
- marsadm join-resource $mydata /dev/vg/$mydata
- marsadm view: wait for UpToDate
- marsadm primary $mydata
- $vmmanager start /dev/mars/$mydata

=> also works with 2 old replicas → 2 new replicas

Example: football.sh in github.com/schoebel/football

MARS Presentation by Thomas Schöbel-Theuer
Planner: produces **stateful plan**

- Complexity = $O(|state|^2)$

Optimizer: produces **stateless actions**
- works like a CONTROLLER LOOP
- similar to Kubernetes
Football Architecture (grey = not yet implemented)

- **Strategy Layer** (orchestration)  
  - x 1
  - early alpha stage

- **Execution Layer** (choreography with locking)  
  - in production!
  - ~ x 1-100

- **Containers / VMs**  
  - ~ x1000

- **Block Layer**

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

---

- pool-optimizer.sh + plugins
- football.sh + plugins
- systemd or cm3
- KVM / qemu
- MARS / DRBD On top of LVM

---

- control path mostly ssh
- data path
- high parallelism degree
- more plugins possible, e.g. libvirt, ...
- another x 2 for geo-redundancy

---

MARS Presentation by Thomas Schöbel-Theuer
Pool-optimizer (early alpha stage)

conceptually almost stateless

MARS Presentation by Thomas Schöbel-Theuer
football.sh (in production with cm3 plugin)
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
  - 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 internal Efficiency project
  - get rid of old hardware
  - Concentrate ~ 7 LXC containers on 1 hypervisor
  - currently >50 „kicks“ per week
    - limited by hardware deployment speed
    - Proprietary Planner (for HW lifecycle)
  - Almost finished: ~70% of ~1800 blades already migrated (mid of January 2019) and mostly shrunk
Best for > 1 PiB of enterprise-critical data
- Example: ShaHoLin (slide3)
- More 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-manual.pdf)
- Architectural **reliability** better than BigCluster with cheaper hw + network!

ask me: decades of experience with enterprise-critical data and long-distance replication
MARS Current Status

- MARS source under GPL + docs:
  github.com/schoebel/mars
  mars-manual.pdf ~ 100 pages

- mars0.1stable productive since 02/2014

- Backbone of the 1&1 geo-redundancy feature

MARS status January 2018:
  > 5800 servers (shared hosting + databases)
  > 2x12 petabyte total
  ~ 10 billions of inodes in > 2500 xfs instances,
    biggest ~ 40 TB
  <= 10 LXC Containers on 1 Hypervisor

- New internal Efficiency project
  - Concentrate more LXC containers on 1 hypervisor
  - New public branch mars0.1b with many new features, e.g. mass-scale clustering, socket bundling, remote device, etc
  - mars0.1b currently in ALPHA stage
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 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
Replication at Block Level vs FS Level

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

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

**Caching Layer**
Page Cache, dentry Cache, ...
1:100 reduction

**Block Layer**
LVM, DRBD / MARS

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

**Potential Cut Point A**
for Distributed System
~ 1.000 Ops / s

**Potential Cut Point B**
for Distributed System
~ 25 Operation Types
~ 100.000 Ops / s

**Potential Cut Point C**
for Distributed System
~ 2 Operation Types (r/w)
~ 1.000 Ops / s

++ replication of VMs for free!

=> Cache Coherence Problem!

MARS Presentation by Thomas Schöbel-Theuer
**DRBD+proxy** (proprietary)

**Application area:**
- Distances: any
- Asynchronously
  - 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 Light** (GPL)

**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 \(\geq 100\)GB in `/mars/` for transaction logfiles
  - dedicated spindle(s) recommended
  - RAID with BBU recommended
- Easy scaling to \(k>2\) nodes
DRBD Host A (primary)  
DRBD Host B (secondary)  
Proxy A'  
Proxy B' (essentially unused)  
Proxy 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)
MARS Data Flow Principle

Host A (primary)

/dev/mars/mydata

Transaction Logger

Temporary Memory Buffer

writeback in background
append
/dev/lv-x/mydata

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

Host B (secondary)

Logfile Replicator

Logfile Applicator

long-distance transfer

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

/dev/lv-x/mydata
Framework Architecture for MARS + future projects

Framework Application Layer
MARS Light, MARS Full, etc

Framework Personalities
XIO = eXtended IO ≈ AIO

Generic Brick Layer
IOP = Instance Oriented Programming
+ AOP = Aspect Oriented Programming

External Software, Cluster Managers, etc

Userspace Interface marsadm

MARS Light

MARS Full

... future Strategy bricks

other future Personalities and their bricks

Generic Bricks

Generic Objects

Generic Aspects