Virtual Petabytes Storage Pools: Agenda

- Storage Architectures → Scalability & Costs
- HOWTO Background Migration of LVs
e.g. for load balancing, HW lifecycle, etc
- Use Cases for Storage Architectures
- Reliability of Storage Architectures
- Flexible MARS Sharding + Cluster-on-Demand
- Current Status / Future Plans
Badly Scaling Architecture: **Big Cluster**

Data already partitioned + isolation needed

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
<th>User 7</th>
<th>User 8</th>
<th>User 9</th>
<th>User 10</th>
<th>User 11</th>
<th>User 12</th>
<th>User 13</th>
<th>User 14</th>
<th>...</th>
<th>User 999999</th>
</tr>
</thead>
</table>

Internet  \( O(n^2) \) REALTIME Access

like cross-bar

Internal Storage (or FS) Network

\( O(n^2) \) REALTIME Access

for geo-redundancy

MARS Presentation by Thomas Schöbel-Theuer
Well-Scaling Architecture: Sharding

Internet O(n*k) ✓

Storage + Frontend 1
Storage + Frontend 2
Storage + Frontend 3
Storage + Frontend 4
Storage + Frontend 5
Storage + Frontend 6
Storage + Frontend 999

+++ traffic shaping possible

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

Smaller Replication Network for Batch Migration O(n)

+++ traffic shaping possible

=> method really scales to petabytes

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

+++ big scale out +++

x 2 for geo-redundancy

MARS Presentation by Thomas Schöbel-Theuer
### HOWTO Background Migration of LVs

#### HOST A (old) VM is running → HOST B (new) has spare space

<table>
<thead>
<tr>
<th>Step</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>lvdisplay /dev/vg/$mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(meanwhile VM is altering data)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$vmmanager stop /dev/mars/$mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>marsadm leave-resource $mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>lvremove $mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>lvcreate -L $size $mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>marsadm join-resource $mydata /dev/vg/$mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>marsadm view: wait for UpToDate</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>marsadm primary $mydata</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$vmmanager start /dev/mars/$mydata</td>
</tr>
</tbody>
</table>

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

Example: tetris.sh in github.com/schoebel/mars/contrib/
Guidelines: Use Cases

Big Cluster
- Objects with non-meaningful keys
- No logical dependencies between objects → failures should not propagate
- No data partitioning possible

Sharding on top of RAID
- Legal requirements (know where the data is)
- Data is already partitioned
- Structured keys (pathnames)
- Recursively structured data with in-place updates, e.g. Block Devices, VMs, ...
- POSIX-complicant FS needed

Beware
- Filesystems on top of spreaded unreliable objects
- Block devices on top of spreaded unreliable objects

Grey Zones
- when artificial partitioning is possible...
- when data is highly volatile
Reliability of Architectures: NODE failures

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

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

need k >= 3 replicas here
Precondition: CPU must not be the bottleneck

Idea: passive LV roles get less CPU

1 datacenter out of 3 may fail

Total Storage: \( x \times 2 \)
Total CPU: \( x \times 1.5 \)
\[ \Rightarrow 1.5 \times O(n) \]

HOWTO flexible CPU assignment \( \Rightarrow \) next slide
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
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)
  > 2x12 petabyte total
  ~ 10 billions of inodes in > 2500 xfs instances, biggest ~ 40 TB

  up to 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
MARS Future Plans

- Automatic load balancing
- Virtual LVM-like Storage + VM pools
- Physically sharded pools

**TBD**
Separate implementation or libvirt / Openstack / Kubernetes plugins … ?

**WIP tetris.sh** +
1&1 clustermanager cm3 and/or systemd and/or libvirt plugin … ?

**Done**
MARS instead of DRBD

**Collaboration sought**
=> Opportunities for other OpenSource projects!
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
- ~ 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!
## Use Cases DRBD+proxy vs MARS Light

<table>
<thead>
<tr>
<th>DRBD+proxy (proprietary)</th>
<th>MARS Light (GPL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application area:</strong></td>
<td></td>
</tr>
<tr>
<td>- Distances: any</td>
<td>- Distances: any ( &gt;&gt;50 km )</td>
</tr>
<tr>
<td>- Aynchronously</td>
<td>- Asynchronously</td>
</tr>
<tr>
<td>- Buffering in RAM</td>
<td>- near-synchronous modes in preparation</td>
</tr>
<tr>
<td>- Unreliable network leads to <strong>frequent re-syncs</strong></td>
<td></td>
</tr>
<tr>
<td>- RAM buffer gets lost</td>
<td>- Tolerates <strong>unreliable network</strong></td>
</tr>
<tr>
<td>- at cost of actuality</td>
<td>- Anytime consistency</td>
</tr>
<tr>
<td>- <strong>Long</strong> inconsistencies during re-sync</td>
<td>- no re-sync</td>
</tr>
<tr>
<td>- Under pressure: <strong>permanent</strong> inconsistency possible</td>
<td>- Under pressure: no inconsistency</td>
</tr>
<tr>
<td>- possibly at cost of actuality</td>
<td>- possibly at cost of actuality</td>
</tr>
<tr>
<td>- High memory overhead</td>
<td>- Needs &gt;= 100GB in <code>/mars/</code> for transaction logfiles</td>
</tr>
<tr>
<td>- Difficult scaling to k&gt;2 nodes</td>
<td>- dedicated spindle(s) recommended</td>
</tr>
<tr>
<td></td>
<td>- RAID with BBU recommended</td>
</tr>
<tr>
<td></td>
<td>- Easy scaling to k&gt;2 nodes</td>
</tr>
</tbody>
</table>

**MARS Presentation by Thomas Schöbel-Theuer**
DRBD+proxy Architectural Challenge

DRBD Host A (primary)

Proxy A'

A != A' possible

Proxy B'

(essentially unused)

DRBD Host B (secondary)

bitmap A

huge RAM buffer

data queue path (several GB buffered)

completion path (commit messages)

bitmap B

sector #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

Temporary Memory Buffer

Transaction Logger

/writeback in background
/append

/dev/lv-x/mydata

Host B (secondary)

Logfile Replicator

long-distance transfer

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

/wdev/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
...

XIO bricks
future Strategy bricks
other future Personalities and their bricks

Generic Bricks
Generic Objects
Generic Aspects