IDIAP COMMUNICATION REPORT

HAVC-II - IDIAP PRIVATE CLOUD (TECHNICAL INSIDE-OUT)

Cédric Dufour

Idiap-Com-01-2015

JULY 2015

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

HAVC-II - Idiap Private Cloud

Technical Inside-Out

– this page is intentionally left blank –

Table of Contents

Index of Tables

– this page is intentionally left blank –

HAVC-II - Idiap Private Cloud

Abstract

Virtualizing resources – servers, storage, networks – is now part of every IT departments life. The benefits of virtualization no longer need to be demonstrated and, when played upon in a large scale, provide both the saddle and spurs with which the *Cloud* mantra has been riding towards its success. This document shall describe how Idiap took advantage of its infrastructure to build its own virtualization farm, which shall (shamelessly) be referred to as *Idiap Private Cloud*.

Introduction

Idiap has been following the trend of virtualizing resources since 2008 and – through the years – developed what became its own virtualization farm, based on common open source solutions and internally dubbed Idiap *High-Availability Virtualization Cluster (HAVC)*.

With the funding opportunity offered by the BEAT (Biometrics Evaluation and Testing) project[1], Idiap was given the means to enlarge its virtualization farm and – after a thorough engineering review – make the best possible use of all its available hardware, to answer both BEAT requirements and Idiap general needs.

Nicknamed Idiap *High-Availability Virtualization Cluster, 2nd generation (HAVC II)*, this internal project – undertaken by Idiap System Group – eventually led to what can now be publicly referred to as *Idiap Private Cloud*.

This document shall cover all the objectives and steps which allowed this project to come to fruition, describing all its aspects, from power supply provisioning to virtual machines commissioning, across processing hardware, storage, network, operating system and software considerations.

1 http://www.beat-eu.org, project funded by the European Commission under the Seventh Framework Programme (FP7)

Overview

In order to start with the general picture, this section will give an overview of Idiap infrastructure, along the policies and historic that are relevant to the HAVC II project.

Processing Hardware

Following a thorough comparison and evaluation of all aspects – features, operations, maintenance, warranty, cost, etc. – of its existing commodity-based processing hardware versus vendor-centric "all-in-one" offerings, Idiap chose in 2011 to migrate its processing resources to *IBM*[2] *BladeCenter (H)* solutions.

Retrospectively, experience has shown that if such solutions do possess some caveats – IBM hardware undoubtedly requires greater knowledge (and patience) to reach configuration objectives – they do allow in the end to lower the overall operational burden (and cost) as well as provide the means to significantly/easily increase the global system performances.

Based on that experience and given the BEAT funding opportunity, Idiap chose in early 2014 to extend its processing hardware base with *IBM FlexSystem* solutions.

The following table details all the processing hardware that has thus been taken advantage of to build Idiap HAVC II:

IBM BladeCenter (N°1 & 2)			
Node Type	*CPU*	*RAM*	*Nodes Qty*
HS22	2x Intel Xeon L5640 → 12 cores	12x PC3L-10600 8GiB → 96GiB	6
HX5	2x Intel Xeon E7-2830 → 16 cores	16x PC3L-8500 8GiB → 128 GiB	10
IBM FlexSystem (N°1 & 2)			
Node Type	*CPU*	*RAM*	*Nodes Qty*
X240	2x Intel Xeon E5-2690v2 → 20 cores	16x PC3-12800 16GiB → 256GiB	8

Table 1: Processing Hardware

24 processing nodes

392 CPU cores

3'904 GiB RAM

Storage Hardware

Historically, Idiap has relied on *NetApp*[3] filers as its main storage resource. Even though competitors alternatives have been analyzed when major new investments were looked into – in particular in response to the BEAT funding opportunity – Idiap has stuck to this original choice.

2 www.ibm.com,"IBM" is a registered trademark owned by International Business Machines Corporation
3 www.netapp.com, "NetApp" is a registered trademark owned by NetApp, Inc.

The BEAT project allowed to extend the existing NetApp resources with new material, the current storage hardware picture now being:

NetApp Filer (N°1)			
Filer Type	*Storage Type*	*Storage Capacity*	*Filer Qty*
FAS3240 (dual) Ontap 8.1 (“seven mode”)	48x 2TB SATA (7.2kRPM)	96TB (raw) / ~52TB (actual)	1
NetApp Filer (N°2)			
Filer Type	*Storage Type*	*Storage Capacity*	*Filer Qty*
FAS3220 (dual) Ontap 8.2 (“cluster mode”)	24x 900GB SAS (10kRPM) 20x 800GB SSD	22TB (raw) / ~10TB (actual) 16TB (raw) / ~11TB (actual)	1

Table 2: Storage Hardware

Each NetApp filer is configured to provide active/active redundancy and load-balancing thanks to its dual “heads”.

2 filers (internally redundant)

~73 TB capacity

(SATA, SAS and SSD)

Those two filers are not strictly devolved to HAVC II. They also fulfill other Idiap storage requirements (such as home, group and project directories, application-dedicated directories, etc.).

Network Hardware

Since the very beginning, Idiap has relied on *HP Procurve*[4] hardware to build its network infrastructure. Even though competitors alternatives have been analyzed when major new investments were looked into, Idiap has stuck to this original choice.

Idiap thus relies on HP Procurve hardware for its core switching fabric, as detailed in the following table:

Core Switching Fabric			
Switch Type	*Available Ports*	*Used Ports (HAVC II)*	*Switch. Qty*
HP E8212zl	44x 10GbE → 440Gb/s (max. 560Gb/s)	8x 10GbE (processing) → 80Gb/s 4x 10GbE (storage) → 40Gb/s	2

Table 3: Core Switching Fabric

Those two *HP Procurve E8212zl* are configured to provide *Distributed Trunking*, thus allowing the uplink of each peer to be split between the two core switch and provide active/active redundancy and load-balancing.

4 www.hp.com, “HP”and “Procurve” are registered trademarks owned by Hewlett-Packard Company

The switching resources of the IBM BladeCenter and FlexSystem chassis – described below - are then connected directly to the core switching fabric (using *Distributed LACP Trunks*):

IBM BladeCenter (N°1 & 2)			
Switch Type	*Available Ports*	*Used Ports*	*Switch. Qty*
BNT 10GbE Virtual Fabric	14x 10GbE (internal) → 140Gb/s 10x 10GbE (uplink) → 100Gb/s	8x 10GbE (internal) → 80Gb/s 2x 10GbE (uplink) → 20Gb/s	4
IBM FlexSystem (N°1 & 2)			
Switch Type	*Available Ports*	*Used Ports*	*Switch. Qty*
IBM EN4093(R)	14x 10GbE (internal) → 140Gb/s 10x 10GbE (uplink) → 100Gb/s	4x 10GbE (internal) → 40Gb/s 2x 10GbE (uplink) → 20Gb/s	4

Table 4: Processing Switching Resources

As well as the NetApp filers:

NetApp Filer (N°1)			
Filer Type	*Available Ports*	*Used Ports*	*Filer. Qty*
FAS3240 (dual)	4x 10GBase-SR → 40Gb/s	4x 10GBase-SR → 40Gb/s	1
NetApp Filer (N°2)			
Filer Type	*Available Ports*	*Used Ports*	*Filer. Qty*
FAS3220 (dual)	4x 10GBase-SR → 40Gb/s	4x 10GBase-SR → 40Gb/s	1

Table 5: Storage Uplink Resources

2 core switches (redundant)

160 Gb/s (processing)

80 Gb/s (storage)

(User) Network Segments

User network traffic is split at Idiap in three main categories and corresponding network segments:

(User) Network Segments	
Segment	*Purpose / Description*
INTRANET	internal services, restricted to contracted users operated by Idiap System Group
DMZ	public services, accessible by anyone (Internet) operated by Idiap System Group
LAB	services not operated by Idiap System Group

Table 6: (User) Network Segments

Network segmentation is implemented using 802.1Q VLANs while traffic policing between segments is enforced by Idiap central (and dual/redundant) firewall.

Other special-purpose network segments do exist and shall be mentioned – later – when relevant to HAVC II.

Uninterruptible Power Supply

Through the years, Idiap has gradually improved its ability to handle facility power outages.

Thanks to its two *APC/MGE*[5] battery-powered UPS (Uninterruptible Power Supply) – which specifications are provided below – Idiap is nowadays able to sustain a >30 minutes power outage.

UPS (N°1 & 2)			
UPS Type	*Available Power*	*Used Power (HAVC II)*	*UPS. Qty*
MGE Galaxy 5000	40kVA	~3kW	2

Table 7: Uninterruptible Power Supply

Those two UPSs are not strictly devolved to HAVC II and also provide the require power backup to other Idiap critical systems. Each UPS is also able to bear the burden of its peer should it fail (with an autonomy reduced by half should a facility power outage happen at the same time).

2 battery-powered UPS (redundant)

40 kVA (capacity)

Coupled with the automatic power-off of uncritical resources (should the facility power outage last longer than 15 minutes), the actual UPS autonomy is larger than the ~30 minutes (at full load) and expected to be closer to ~60 minutes.

Operating System and Software Suite

Though its Unix history had it venture on the soil of various Unix-like operating systems, Idiap nowadays rely solely on the *Debian*[6] *Linux*[7] (64-bit) distribution to power its servers infrastructure.

Favoring stability and security over leading-edginess of open source software – as far as servers are concerned – Idiap relies in particular on the *Debian/Stable* branch, also known as *Debian/Wheezy* at the time of writing.

Since 2011, Idiap has been virtualizing its servers resources using the open source virtualization and high-availability software described below, all readily available as (appropriately bundled and pre-configured) Debian packages:

Debian/Wheezy 64-bit		
Software	*Internet Link*	*Description / Purpose*
KVM	www.linux-kvm.org	hardware-accelerated virtualization
QEMU	www.qemu.org	x86 hardware emulation/virtualization
libvirt	www.libvirt.org	virtualization (abstraction) API
Corosync	corosync.github.io/corosync	group (cluster) communication system
Pacemaker	www.clusterlabs.org	high-availability resource manager

Table 8: Operating System and Software Suite

5 www.apc.com, “APC” and “MGE” are registered trademarks owned by Schneider Electric
6 www.debian.org, “Debian” is a registered trademark owned by Software in the Public Interest, Inc.
7 www.linuxfoundation.org, “Linux” is a registered trademark owned by Linux Torvalds

Linux Debian/Wheezy (64-bit)
QEMU/KVM + libvirt (virtualization)
Corosync/Pacemaker (high-availability)

Keep It Simple, Stupid (KISS)

There are several alternative software suites that target large scale virtualization – or so-called *Cloud* enablement stacks – relying on (and somewhat "hiding") the QEMU/KVM(/libvirt) combo.

Idiap has chosen to **stick to the most basic approach**, both based on its past experience with it and for the sake of its (relative) simplicity.

This approach allows in particular to easily bypass the (Corosync/Pacemaker) high-availability layer and recover the lowest possible control on the (QEMU/KVM) virtualization layer, if (when!) needs be.

Summary

The available hardware and software used can be summarized as:

Available Hardware	
Type	***Summary***
Processing	**24 nodes, 392 CPU cores, 3904GiB RAM**
Storage	**~73TB capacity (SATA, SAS, SSD)**
Network	**160Gb/s (processing) 80Gb/s (storage)**
Power	**full redundancy (against UPS failure and/or facility outage)**
Software Used	
Type	***Summary***
OS	**Linux Debian/Wheezy (64-bit)**
Virtualization	**QEMU/KVM + libvirt**
High-Availability	**Corosync/Pacemaker**

Table 9: Summary of Available Hardware and Software Used

Challenges

Sensibly taking advantage of the available hardware to obtain optimal performances without sacrificing data and operational security introduces many challenges, which shall be covered in this section.

High-Availability Traffic and Live Migration

Achieving high-availability in a computer system environment implies using a software stack that shall:

- gather several processing hosts – or *nodes* – into a logical group – or *cluster* – for the sake of sharing/exchanging their ability to host services – or *resources* – that must remain available “no matter what”
- monitor the “health” of all nodes, resources and the overall cluster
- take appropriate actions should this monitoring fail or spawn unwanted results, entirely automatically (as in “without any human intervention”)

This implies messages – commonly referred to as *high-availability traffic* – must be exchanged within the cluster in order to keep track of its status, while resources must be *migrated* from one node to another in case of problems.

High-Availability Traffic

As we just saw, the high-availability cluster must maintain immediate and reliable communication between all its nodes at all time, to keep track of its own health/status.

In other words, the (low-bandwidth) traffic that corresponds to the cluster health/status messages must not be lost or delayed (further than the acceptable/configured limits).

Live Migration

When speaking of virtualization, *live migration* is the very neat feature that allows one virtual machine to be moved from one physical host to another without – apparently (to the human eye) – stopping its operations.

This is achieved by – simply put – synchronizing the full *state* (CPU, RAM, storage) between the “stopping” virtual machine (on one node) and its “starting” peer (on another node).

The problem that live migration raises is the performance – more precisely the bandwidth – of the network through which the state data shall be transmitted. With virtual machines which state may extend to several gigabytes of data, the synchronization can take several tens of seconds, if not minutes.

Unfortunately, high-availability is about avoiding (or reducing as much as possible) unavailability. In this context, several minutes unavailability are unacceptable. Health monitors (shortly mentioned in the previous chapter) will timeout way before minutes passed, prompting the cluster for actions. Virtual machines state synchronization must thus be achieved as fast as possible (or within acceptable/configured limits).

High Performance Network

The network shall be set up such as to prioritize high-availability traffic and provide maximum performances for live migration.

Quality of Service (QoS)

The previous chapter already mentioned two stringent constraints to be taken in consideration when designing the network topology on which to build HAVC II.

More generally speaking, unused costly resources are a waste of money. In order to use resources as efficiently as possible, they ought to be shared as much as possible (between users, applications, purposes, etc.), in order to suppress the boundaries (limits) that would prevent one use-case to take full advantage of one resource when it would otherwise remain idle.

The problem raised by sharing resources is the potential abuse and starving of resources by one (or more) use-case(s) to the detriment of the others. In other words, shared resources starving must be prevented.

Use cases usually falls into easily identifiable categories: INTRANET vs LAB services, high-availability vs peer-to-peer traffic, etc. This is where *Quality of Service (QoS)* technologies come handy, by allowing to prioritize resources usage according to the identified categories.

A global Quality of Service (QoS) policy shall be devised and implemented throughout Idiap network, in respects with identified use-cases categories.

Network Adapters Bonding

All the processing nodes available for the HAVC II project are equipped with two *Network Interface Cards (NICs)*. As was explained in the previous chapter, those two network adapters can be more efficiently used if *bonded* together – to provide a shared double-capacity link – rather than each dedicated to a specific purpose.

Network adapters bonding also provides redundancy (and thus higher-availability), the resulting network link remaining functional should one the network adapter (or uplink switch) fail.

The Linux kernel offers various bonding modes, from simple *active-backup* teaming to advanced active/active *balance-alb* or *802.3ad* LACP trunking.

Each mode unfortunately has its pros and cons:

- active-backup:
  [pros] no specific switch configuration, no ARP issues
  [cons] only half the bandwidth available
- balance-alb:
  [pros] no specific switch configuration, maximal bandwidth
  [cons] ARP issues
- 802.3ad LACP:
  [pros] maximal bandwidth, no ARP issues
  [cons] requires specific switch configuration

Unfortunately, as will become obvious in the next section, the best choice of network adapter bonding was not easy to make.

The network adapter bonding mode shall be chosen such as to provide the best compromise to match the overall HAVC II requirements.

High-Availability Cluster Size

The Corosync/Pacemaker combo has been known and used for years in the open source world to provide high-availability for “small” cluster scenarios (“small” as in typically two-to-eight nodes, hosting a few tens of resources).

Idiap will to extend its high-availability cluster to more than 20 nodes and several hundreds resources is – still – not something that is common among Corosync/Pacemaker community (as was confirmed by Andrew Beekhof himself, as head-developer of Pacemaker).

As became obvious while deploying HAVC II based on the stock Pacemaker version (1.1.7) available in Debian/Wheezy, dealing with several hundreds resources on close to 20 nodes was leading to major performances issues and high-availability hiccups.

Corosync/Pacemaker shall be set up such as to allow a high-availability cluster spanning several tens of nodes and several hundreds of resources.

Storage Outage Resilience

As one may recall from the previous section, HAVC II relies entirely on NetApp filers for its storage backend.

More to the point, following tests conducted in 2013 (which results are beyond the scope of this document[8]), Idiap even chose to rely entirely on NetApp NFS exports (rather than block-level LUNs) to host virtual machines images.

While both NetApp filers are equipped with two “head” nodes configured for active/active high-availability, experience has shown that NFS would freeze for several tens of seconds in case of a fail-over, thus resulting in apparent storage outage and node fencing actions in the HAVC (II) cluster, despite the fact that NFS operations would resume totally healthily once the fail-over was complete.

The QEMU/KVM virtualization layer shall be configured such as to sustain several-minutes NFS storage outage gracefully (that is, without having the high-availability loose its senses and starting to fence nodes off).

8 shortly put: NetApp NFS performances were comparable to block-level LUNs performances, thus saving the trouble/complexity of the iSCSI approach

Node Fencing (STONITH)

We have already and shortly covered in the first chapter of this section what a high-availability software stack is and how it shall handle its hosted resources to guarantee their availability.

Thou shall not split brain!

One particular situation that may arise in a high-availability setup is a network failure that would prevent one or more nodes to communicate with the rest of the cluster.

For the sake of maintaining the availability of the resources hosted by those lost nodes, the cluster must (re)start those resources on the remaining healthy nodes. But it can not do so until it has the absolute guarantee that the affected resources are stopped on the faulty nodes. Failure to enforce this rule would result in a potential *split-brain* situation (one body controlled by two separate half-brains, a metaphor to underline the fact that anything awry may then happen).

Since network communication is lost, the only way the cluster can obtain that guarantee is by *Shooting The Other Node In The Head (STONITH)*, that is – in the context of computer systems – cutting off the power of the faulty node (or, simply put, “pull the plug”).

This action is also referred to as node *fencing* (in reference to putting the node apart, behind a safe fence).

Management Modules to the Rescue

HAVC II processing nodes being part of fully integrated management solutions in the form of the IBM BladeCenter or FlexSystem chassis, node fencing can be achieved thanks to the remote power control ability provided by the chassis management modules.

Both IBM BladeCenter *Advanced Management Module (AMM)* and IBM FlexSystem *Chassis Management Module (CMM)* provide a *SNMP (Simple Network Management Protocol)* interface, which can be used to control the processing nodes power via appropriate scripts.

Scripts – also known as *STONITH plugins/modules* – shall be written such as to allow Pacemaker to perform node fencing via IBM BladeCenter AMM and IBM FlexSystem CMM.

Into Details

Now that the hardware context and most challenging issues have been covered, this section will go into the implementation details of Idiap High Availability Virtualization Cluster, 2nd generation (HAVC II).

Network Topology

In many high-availability setup, the *cluster* network is entirely separated from the *access* network, using dedicated hardware: network adapters, links and – in the most extreme scenario – switches. While this topology offers guaranteed performances for the high-availability and live migration traffic, it significantly increases costs. Also, dedicating the hardware for a specific purpose makes it much more difficult – if not impossible – to achieve redundancy by using hardware dedicated to another purpose.

Physical Topology

For HAVC II, we chose not to separate the cluster network from the access network and to use all the available hardware to setup a **single physical network**.

Since **every element in this network is duplicated** – network adapters, links and switches – this approach allowed us to provide **total redundancy** from the network point of view.

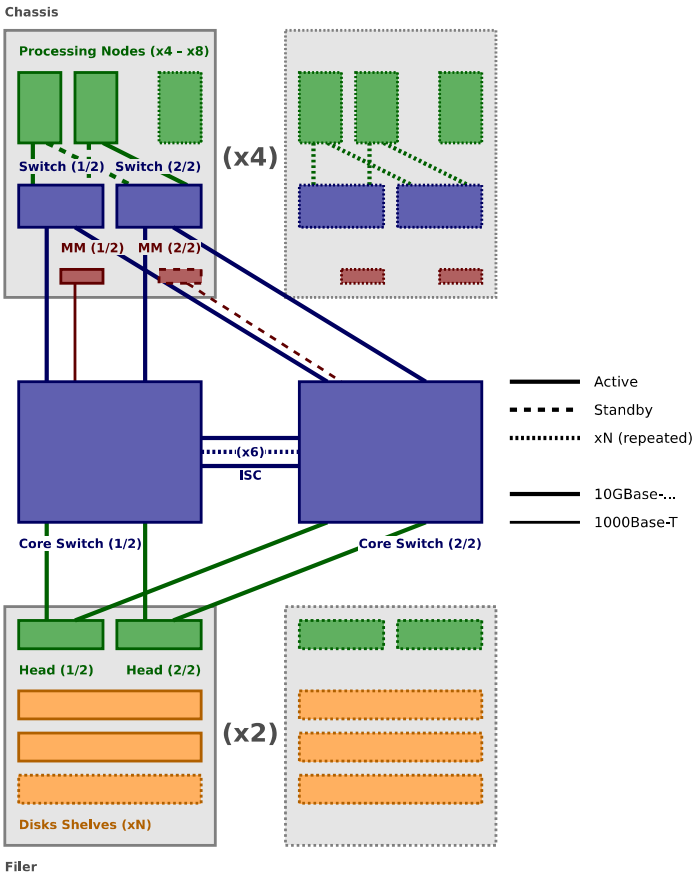

Illustration 1: (Network) Physical Topology

This topology provides **active/active redundancy and load-balancing**, save for two exceptions:

- **Processing Nodes**: as shall be explained in a following chapter, processing nodes dual network adapters were bonded using ***active-backup*** mode (with each interface connected to one of the two chassis switches). During non-degraded operations, each server has its active interface alternated between the two switches, thus balancing the load optimally. ***Uplink Failure Detection*** has been enabled on the chassis switches, thus guaranteeing that the appropriate processing node interface is active in case the switch uplink fails.
- **Management Modules**: IBM management modules can only be operated in **active/standby** mode. Moreover, each module only has a single network interface. In order to guarantee the availability of the management module: 1. each module has been connected – directly – to one of the two core switches; 2. ***Uplink Failure Detection*** has been enabled on the management modules, thus guaranteeing that the appropriate module is active in case one of the core switch fails.

Logical Segments/Categories

The various type of traffic – high-availability, live migration, user (access), etc. - are segregated using **802.1Q VLANs** and **TCP/UDP port-based ACLs**, thus providing **logical separation** (instead of physical separation).

(Network) Logical Segments/Categories		
Category/Segment	*Type*	*Description / Purpose*
Cluster (internal)	802.1Q VLAN	Cluster internal network Live migration traffic
HA	TCP/UDP port	Corosync group communications
Management	802.1Q VLAN	Management network
Backend (hypervisor, storage)	802.1Q VLAN	Cluster nodes (hypervisors) access network Storage (disk images) access network
INTRANET (VM, user)	802.1Q VLAN	Internal services, restricted to contracted users Operated by Idiap System Group
DMZ (VM, user)	802.1Q VLAN	Public services, accessible by anyone (Internet) Operated by Idiap System Group
LAB (VM, user)	802.1Q VLAN	Services not operated by Idiap System Group

Table 10: (Network) Logical Segments/Categories

Quality of Service (QoS)

Network Quality of Service (QoS)

802.1p priorities and Guaranteed Minimum Bandwidth (GMB) have been implemented throughout Idiap network and hosts in order to obtain **optimal performances** during normal operations and **guaranteed minimum performances** in case of network overload.

802.1p priorities

802.1p priorities are a simple mechanism which allow to associate one among **eight priority levels** to network packets. Based on these priorities, packets are then “routed” towards the network host **(numbered) *egress* queues**, where a **queue with a higher number usually has strict priority over a queue with a lower number**.

802.1p Priorities and Queues		
Category/Segment	*802.1p*	*Queue*
HA	**7 (highest)**	**7**
Management	**6 (very high)**	**6**
Cluster (internal)	**5 (high)**	**5**
Backend (hypervisor, storage)	**4 (medium-high)**	**4**
INTRANET (VM, user)	**3 (medium-low)**	**3**
DMZ (VM, user)	**0 (low, default)**	**2**
LAB (VM, user)	**2 (very low)**	**1**

Table 11: (Network) 802.1p Priorities and Queues

One can see here:

- high-availability traffic has been attributed the highest priority; we don't want the high-availability stack – Corosync/Pacemaker – to loose its bearing and start behaving berserk
- then management traffic; system administrators should always be able to connect to the system management interfaces
- followed by (mostly) live migration traffic; live migration does not happen often but when it does, we want it to proceed and complete as fast as possible
- then (mostly) virtual machines disk images (storage) traffic; no point in prioritizing user traffic if the underlying backend fails to keep up
- and finally user traffic, according to its importance

Unfortunately, without additional bandwidth control, this scenario is prone to the so-called **queue starvation** issue, where traffic in a lower-numbered queue is blocked and kept waiting until no more traffic has to be delivered from higher-numbered queues.

Guaranteed Minimum Bandwidth (GMB)

In order to prevent queue starvation, *Guaranteed Minimum Bandwidth (GMB)* has been added to the 802.1p priorities. This mechanism is used to guarantee that the minimum configured bandwidth for a queue is always honored, no matter what happens in higher-numbered queues.

(Network) Guaranteed Minimum Bandwidth (GMB)		
Category/Segment	*Queue*	*GMB*
HA	**7**	**strict priority**
Management	**6**	**10%**
Cluster (internal)	**5**	**25%**
Cluster (hypervisor, storage)	**4**	**25%**
INTRANET (VM, user)	**3**	**20%**
DMZ (VM, user)	**2**	**10%**
LAB (VM, user)	**1**	**5%**
(other traffic)	*0*	*5%*

Table 12: (Network) Guaranteed Minimum Bandwidth (GMB)

Guaranteed Minimum Bandwidth vs. Rate Limiting

Another way to prevent queue starvation is to apply *Rate Limiting* to each traffic category/segment, stating that its flow shall not exceed – no matter what – the configured maximum bandwidth.

The major drawback of this technique lies with the fact that a given category/segment may be throttled down unnecessarily, when no higher-priority traffic is competing for the available hardware resources.

Shortly put, **Rate Limiting is a waste of hardware resources**, while Guaranteed Minimum Bandwidth makes optimal use of it.

egress vs. igress traffic control

One should first note that most network hosts implement buffering (queues) only for *egress* (output) traffic.

Given this situation, *igress* (input) traffic control presents the major drawback of dropping packets rather than delaying them (by keeping them in their attributed buffer/queue). **Dropping packets should be avoided by all means**, since it prompts the sending party to resend lost packets – according to its network congestion algorithm – and contributes to increase network load and latencies.

Of course, when network load is too high, dropped packets will occur nonetheless, due to buffers overflow.

Implementation

The network quality of service policy just described must be implemented on all network hosts is order to be fully efficient. Given the context of HAVC II, it must thus be implemented on:

- **HP Procurve** core switches; the required features are readily available through "*qos ...*" and "*interface ... bandwidth-min*" CLI commands
- **IBM (BNT)** chassis switches; the required features are readily available once enabled the switch *Converged Enhanced Ethernet (CEE)* mode and through "*/c/cee/global/ets/pg*", "*/c/acl/acl*" and "*/c/port INT.../acl/add*" CLI commands
- **Linux** (processing nodes) hosts: the required features are readily available thanks to the "*tc*" (traffic control) utility

Storage Quality of Service (QoS)

For the same reasons that drove us to setup a single physical network, we have chosen to take advantage of available storage hardware as a single generic-use backend rather than dedicate specific hardware for one purpose or another.

Storage performances considerations are identical to network ones, which can be addressed with strategies and techniques that exploit the same ideas.

NetApp FlexShare Priorities

NetApp filers running Ontap software in "seven mode" offer ***FlexShare Priorities*** to address **storage prioritization at the volume level**.

(Storage) NetApp FlexShare Priorities		
Volume	*Priority*	*Description / Purpose*
/vol/havc_hypervisor	**90**	HAVC II configuration files and scripts
/vol/havc_intranet	**70**	↔ INTRANET (user) segment
/vol/havc_dmz	**50**	↔ DMZ (user) segment
/vol/havc_lab	**30**	↔ LAB (user) segment

Table 13: (Storage) NetApp FlexShare Priorities

NetApp Ontap "seven mode" vs. "cluster mode"

NetApp Ontap running in "seven mode" provides FlexShare priorities to address storage performances prioritization issues, much the same way as network 802.1p and Guaranteed Minimum Bandwidth (GMB).

Very unfortunately, **NetApp Ontap running in "cluster mode"** does not support FlexShare priorities and **only support storage Quality of Service (QoS) through Rate Limiting** (storage operations/bandwidth).

The same critic that was made of Rate Limiting in the network context applies to storage, namely its resulting in a **waste of resources**.

Software Stack

The following illustration details how the various software elements introduced in the first section interact with each other and their surrounding network environment:

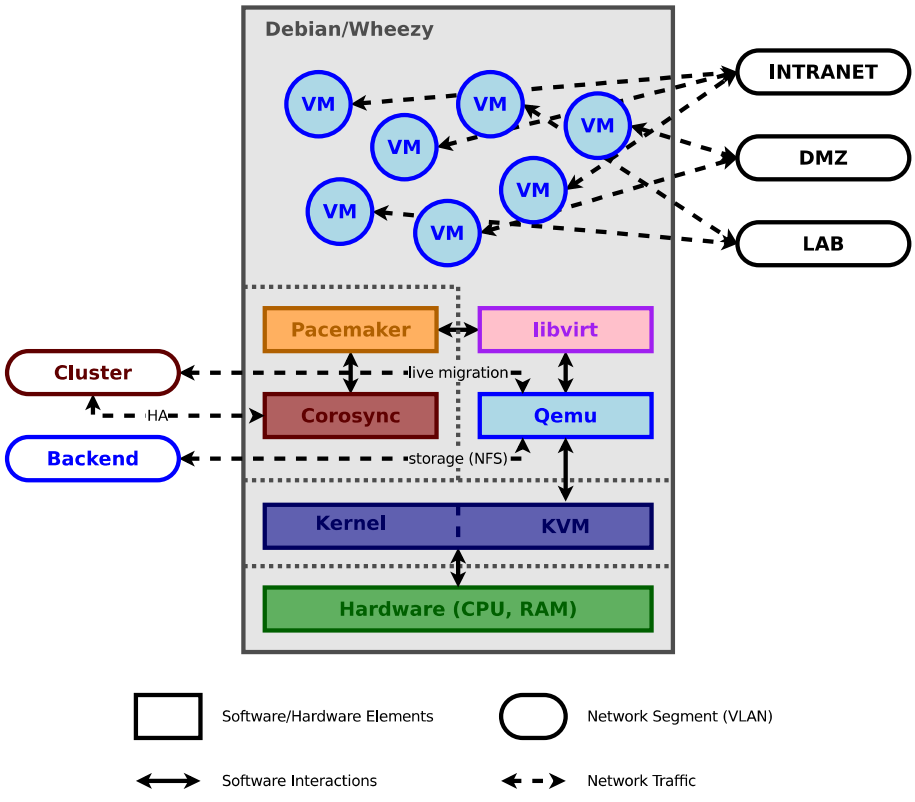

Illustration 2: Software Stack (Interactions and Network)

Network Configuration

In order to provide the network connections required by the various elements, a rather extended network setup had to be configured, keeping in mind the various aspects it shall cover.

Network Bonding

Each processing node being equipped with two network adapters, they have to be bonded together to fit within our “single physical network” topology.

The Linux kernel offers several mode to achieve network bonding, the one relevant to our setup being *active-backup*, *balance-alb* and *802.3ad LACP*. The pros and cons of each mode has already been covered in the previous section. However...

Each node network adapter being connected to each of the two chassis switch, 802.3ad LACP bonding requires that the two switches be *stacked* together. While IBM/BNT switches do support stacking, several features are no longer available when it is enabled. *Converged Enhanced Ethernet (CEE)* is unfortunately among them. Since our setup requires CEE to achieve network Quality of Service (QoS), **802.3ad LACP bonding is not a possible option**.

The active/active bonding mode that remains is balance-alb, which stands for “*Advanced (RX/TX) Load-Balancing*”. This method relies on IPv4 *Address Resolution Protocol (ARP)* and IPv6 *Network Discovery Protocol (NDP)* mangling, such as to appropriately advertise one of the two network adapters MAC address to each external network party. Unfortunately again, when coupled with the bridging setup required by Qemu/KVM, this mode sometimes behaves erroneously and advertises the wrong MAC address to a network party, resulting in network disruption. This bug is present even in the latest backported kernel available for Debian/Wheezy (version 3.14). Thus, **balance-alb bonding is not a viable option**.

The only mode that remains is thus active-backup, where only one of the two network adapters is active and available for network traffic. In case its link goes down, the backup network adapter will then become the active one and take over all the traffic. Coupled with IBM/BNT “*Uplink Failure Detection*” mechanism, **active-backup bonding provides edge-to-core redundancy, though with only half of the potential performances**. Considering the uplink bandwidth of each chassis switch (namely, 20Gb/s), this reduction of performances is leveraged by the fact that during balanced operations (throughout all processing nodes), the uplink is not able to sustain the full network load (of all processing nodes) anyway.

Linux Network Adapters Bonding

802.3ad LACP = no go!

balance-alb = no go!

active-backup = oh well...

Traffic Control

As we have seen, HAVC II relies on 802.1p-like priorities and Guaranteed Minimum Bandwidth (GMB) to achieve the required network Quality of Service (QoS). While this takes only a few configuration lines on HP or IBM/BNT switches, it is much more complex to implement on a Linux box.

This is where the *tc (traffic control)* utility comes into play, allowing the administrator to modify the network adapters *queuing discipline (qdisk)* and replicate the switches setup with a mix of *PRIO* prioritizer, *HTB (Hierarchical Token Bucket)* traffic shapers and *SFQ (Stochastic Fairness Queuing)* queues.

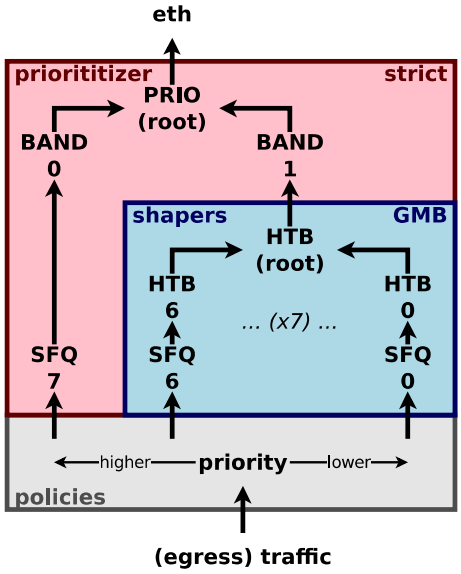

Illustration 3: (Linux) Traffic Control

The actual implementation (script) to achieve this scenario can be found in the final section of this document.

Corosync/Pacemaker

The Right Version

The final purpose of HAVC II is to host a **few hundreds virtual machines**, corresponding to as many high-availability resources.

While commissioning the system with Debian/Wheezy default version of Pacemaker – namely 1.1.7 plus a few patches – it became obvious that Pacemaker had a hard time managing the large quantity of resources, with handling of the growing XML-based *Cluster Information Base (CIB)* resulting in 100% CPU load for several minutes (during which high-availability could not be considered as guaranteed).

Fortunately, the just released *Pacemaker 1.1.12*[9] brought a bunch of fixes and new features, including – specifically – a significant improvement in regards with the CIB: "*Thanks to a new algorithm, the CIB is now two orders of magnitude faster*".

After **backporting Pacemaker 1.1.12 to Debian/Wheezy** – along *libqb 0.17.0*, *corosync 1.4.7* and *crmsh 1.2.6* dependencies – it appeared the new CIB code did hold its promises, allowing our ~400-resources cluster to work back as a charm.

Corosync/Pacemaker for Large Cluster (Rule N°1)

Only Pacemaker 1.1.12 (or later) will do!

LibvirtQemu Custom Resource Agent

Even though Pacemaker 1.1.12 appears to definitely fix the CIB processing issue we encountered, we thought we'd better try to reduce the quantity of HA resources by merging multiple resources *primitives* (per *group*) into as few primitives as possible, ideally into a single one.

9 https://github.com/ClusterLabs/pacemaker/releases/tag/Pacemaker-1.1.12

The rationale for this is two-fold:

- **limit CIB size**: each primitive corresponds to XML code in the CIB; the fewer primitives, the smaller the CIB size
- **limit resources operations**: each resource triggers specific operations throughout the various phase of its high-availability life; the fewer the resources, the fewer those operations (and resulting latencies)

Historically, Idiap had been managing its virtual machines resources thanks to the default *VirtualDomain* and *MailTo* resource agents (and corresponding primitives), the latter being used solely to have informational status e-mail sent when virtual machines were being started or stopped. Given our new resources/primitives requirement, those two could conveniently be merged as a single resource/primitive for each virtual machine.

We thus created a **custom *LibvirtQemu* resource agent, which merges the VirtualDomain and MailTo resource agents into a single one**:

- simplifying its inner structure by targeting Qemu/KVM specifically (rather than all potential Linux virtualization technologies)
- merging and extending the support for informational e-mail messages, now giving more detailed information about what happens at the virtualization level (graceful vs. forced shutdown, live migration, etc.)
- allowing to reduce by half the part of the CIB used for virtual machines configuration and status tracking

Corosync/Pacemaker for Large Cluster (Rule N°2)

Avoid creating multiple-primitives resources (group)!

Create your own custom *all-in-one* resource agent(s)!

The actual script corresponding to Idiap custom LibvirtQemu resource agent can be found in the final section of this document.

SNMP Stonith Plugins

Allowing fencing to be achieved via IBM chassis management modules required that we wrote **scripts** that matched the **Stonith plugins semantic** and appropriately interacted with the modules via **SNMP(v1)**.

STONITH with IBM Blade Center and Flex System chassis...

... must be implemented with custom (SNMP-wrapping) scripts!

The actual scripts allowing to remotely control IBM Blade Center and Flex System chassis can be found in the final section of this document.

Libvirt and Qemu/KVM

Though Debian/Wheezy stock **libvirt** version – 0.9.12.3 – did not give us any reason to complain, we opted to use the available backported version – **1.2.4** – to anticipate potential unforeseen bugs and ease future update.

On the other hand, **Qemu/KVM** was kept at its default Debian/Wheezy version – **1.1.2** – in order to benefit from the thorough security watch and fast responses from the Debian security team.

Local Configuration and Scripts

In order to prevent the LibvirtQemu (Pacemaker/HA) resource agent or STONITH plugins to choke in case of network storage issues (failure or, more likely, performances degradation) – resulting in (highly undesirable) node fencings – all configuration and scripting dependencies must be kept on each node **local file system**.

> **All configuration files and scripts...**
>
> **... must be stored on each node local file system!**

The administrative burden of this requirement has been leveraged by using a shared network repository where all files are actually maintained, before being *rsync*-ed to the local file system of all cluster nodes.

Disk Errors Policies

As mentioned in the previous section, Qemu/KVM ought to sustain NFS disruptions gracefully – that is by just waiting for NFS to be accessible again rather than triggering any Pacemaker reaction – knowing that such disruptions are only transient, given the high-availability of the NFS service itself.

Assuming the NFS backend is absolutely fail-safe, this could be achieved by configuring Qemu/KVM to *ignore* read and write errors (*rerror*/*werror*).

Unfortunately, even clustered NetApp NFS filers are error-prone, especially given human error sources (e.g. underestimated storage capacity...). The safest setting is thus to use Qemu/KVM *enospc* setting, resulting in Qemu/KVM virtual machines being paused in case storage capacity gets exhausted and other errors being reported to the guests. NFS disruptions are thus reported as disk I/O timeouts in the guests, which are usually handled gracefully enough (save for ad-hoc messages being logged).

In libvirt, this setting corresponds to the ***error_policy='enospace'*** attribute of each disk's driver (XML) element.

> **In order to sustain NFS disruptions as gracefully as possible, Qemu/KVM disks...**
>
> **... must be configured to *enospc* read/write errors (*rerror*/*werror*)!**

Live Migration

Successful live migration of Qemu/KVM virtual machines is tightly dependent on stateful disk operations. In other words, disk operations must not resume on the destination host – at all cost! - until the source host is completely done with the disk image(s) and all writes successfully committed to the storage backend. Failure to honor this requirements immediately results in disk/data corruption.

While **Qemu/KVM live migration** of NFS-backed virtual machines works flawlessly, we have discovered that it **fails miserably – with immediate disk/data corruption – if one uses a (local) symlink to the actual NFS-backed disk image**. We suspect this has to do with file locking issues (bugs?), causing Qemu/KVM to become unable to detect when the handover of disk operations can be safely achieved.

In order to prevent disk/data corruption during live migration...

... NFS-backed Qemu/KVM disk devices must NOT use symlinks!

System Administration

Configuring and enabling resources – virtual machines – in HAVC II comes down to writing the proper **XML configuration files for libvirt and Pacemaker** before feeding them to Pacemaker using the **cibadmin** command.

In order to ease and streamline this step, we haven written templates and scripts that eventually allow to bring down the time required to commission a new virtual machine – provided its disk image is available and the DNS/DHCP are already configured – in a few seconds:

- **havc-config-host** ...
- **havc-config-libvirt** ...
- **havc-config-pacemaker** ...
- **havc-config-hardware** ...
- **havc-enable** ...

These scripts are all invoked from our **central administration server**, which is granted the proper permissions to access, configure and control the cluster and its nodes, thus further easing all administrative tasks.

The actual system configuration and administration scripts can be found in the final section of this document.

Conclusion

Using open source software and thoughtful engineering, Idiap has been able to implement a large scale virtualization solution, prioritizing the most efficient use possible of available resources, without sacrificing the service level requirements of the various service categories, and guaranteeing the highest possible availability of the setup.

Thanks to readily available and (Debian) packaged software components, this setup is easy both to implement and maintain, thus minimizing its operational cost.

Having reached its production phase, Idiap High-Availability Virtualization Cluster, 2nd generation (HAVC II) – also known as Idiap Private Cloud – nowadays hosts 200+ virtual machines, one half being general-purpose servers and the other half computation nodes for Idiap computation grid and the BEAT platform.

Acknowledgments

Foremost, for its making the entire project financially possible: the Biometrics Evaluation and Testing (BEAT) project, European FP7 grant n.284989

For their utter patience and support throughout its implementation: Frank Formaz (Idiap System Group manager), Norbert Crettol (my fellow sysadmin), Bastien Crettol (and his acute English-reader eye) and Louis-Marie Plumel (and his apt covering of my blunders when users came screaming to Idiap Helpdesk).

For his fast and acute help: Andrew Beekhof, head developer of the Pacemaker project.

And all the enthusiasts that allow open source software to be so cool to work with.

Cédric Dufour
Martigny – October 24th, 2014

Annexes

This section shall provide all the actual configuration files and scripts relevant to the HAVC II setup. Be careful, here be dark marshes, evil spirits and fiery dragons!

NOTE: actual IP addresses, network masks, VLAN IDs, etc. have been replaced with sample ones.

Network Configuration

/etc/network/interfaces

```
# Loopback
auto lo
iface lo inet loopback

# Interfaces trunk ('active-backup' bond)
auto bond0
# ... primary physical interface
iface eth0 inet manual
  bond-master bond0
# ... secondary physical interface
iface eth1 inet manual
  bond-master bond0
# ... bond
iface bond0 inet static
  # ... parameters
  bond-mode active-backup
  bond-miimon 100
  bond-updelay 3500
  bond-downdelay 500
  # ... slaves
  bond-slaves none
  post-up ifup eth0
  post-up ifup eth1
  # ... IP settings
  address 192.168.3.101
  netmask 255.255.255.0
  gateway 192.168.3.1
```

```
  # ... dependencies
  post-up ifup vlan2
  post-up ifup vlan4
  post-up ifup vlan5
  post-up ifup vlan6
  pre-down ifdown vlan2
  pre-down ifdown vlan4
  pre-down ifdown vlan5
  pre-down ifdown vlan6

# VLANs
# ... 2 (HA)
iface vlan2 inet static
  vlan_raw_device bond0
  # ... IP settings
  address 192.168.2.101
  netmask 255.255.255.0
# ... 4 (INTRANET)
iface vlan4 inet manual
  vlan_raw_device bond0
  post-up ifup br0
# ... 5 (DMZ)
iface vlan5 inet manual
  vlan_raw_device bond0
  post-up ifup br1
# ... 6 (LAB)
iface vlan6 inet manual
  vlan_raw_device bond0
  post-up ifup br2

# Bridge interfaces (for KVM)
# ... VLAN 4 (INTRANET)
iface br0 inet manual
  bridge-ports vlan4
  bridge-maxwait 0
  bridge-stp off
  bridge-fd 0
# ... VLAN 5 (DMZ)
iface br1 inet manual
  bridge-ports vlan5
  bridge-maxwait 0
  bridge-stp off
  bridge-fd 0
# ... VLAN 6 (LAB)
iface br2 inet manual
  bridge-ports vlan6
  bridge-maxwait 0
  bridge-stp off
```

```
  bridge-fd 0
```

Linux Traffic Control

/etc/init.d/traffic-control

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Traffic control start/stop script

# NOTES:
# A. The 'prio' qdisc implements *strict* priority queuing, among the specified
#    "bands" (classes), the lower bands being serviced first, no matter what
# B. The 'htb' qdisc implements *guaranteed minimum bandwidth + priority*
#    queuing, knowing that:
#     1. the kernel "services" (dequeues traffic) from the root HTB class, which
#        *recursively* services its "leaf" sub-classes based on their 'prio'
#        order.
#     2. each of the "leaf" class is serviced unobstrusively as long as its
#        corresponding traffic flow is below the specified 'rate'
#     3. the "remaining" bandwidth ('rate' - actual) can be "lended to" other
#        classes, who can "borrow" it should they need to send traffic above
#        their allocated 'rate' (but below their specified 'ceil')
#     4. "remaining" bandwidth is used to service classes *in order* of their
#        specified 'prio'(rity), lowest first. If several classes have the same
#        'prio' they get a 'rate'-ratioed share of it.
#     5. the actual bandwidth of each class can never excess its specified
#        'ceil'.
# C. The 'sfq' qdisc allows to service traffic more "fairly" (within *each*
#    associated traffic class), than the 'pfifo' qdisc would, by splitting the
#    traffic into "flows" (a flow corresspong ~ to a TCP connection), and
#    servicing each flow in a round-robin fashion. In scenario of congestion,
#    this prevents low-bandwidth flows to be delayed too long by high-bandwidth
#    ones.
# D. TC 'filter' *must* be associated to the qdisk to which 'flowid'(s) they
#    classify the traffic to.

### BEGIN INIT INFO
# Provides:          traffic-control
# Required-Start:    $network
# Required-Stop:     $network
# Default-Start:
# Default-Stop:
```

```
# Short-Description: Traffic control (shaping) configuration script
### END INIT INFO


## Usage
[ $# -lt 1 -o "${1##*-}" == 'help' ] && cat << EOF && exit 2
USAGE: ${0##*/} {start|stop|restart|status}

SYNOPSIS:
  Implement traffic control, similarly to switches-implemented
  Quality-of-Service (QoS):
   - 1 strict priority queue
   - 7 Guaranteed Minimum Bandwith (GMB) priority queues
   - QoS priorities being mapped to queues/bandwidth as:
           qos:   1   2   0   3   4   5   6   7
         queue:   0   1   2   3   4   5   6   7
     bandwidth:  5%  5% 10% 20% 25% 25% 10% strict
EOF

# Arguments
TC_ACTION="${1}"

# Parameters
# ... from configuration
TC_AUTO='yes'
TC_IFACES=( 'eth0' )
TC_RATES=( 1000000 )  # kbit
TC_DEFQ=2
[ -e /etc/default/traffic-control ] && . /etc/default/traffic-control
[ "${TC_AUTO}" != 'yes' ] && exit 0
# ... queues
q_qos=( 1 2 0 3 4 5 6 7 )
q_bw=( 5 5 10 20 25 25 10 strict )  # percent (sum=100)

# Resources
# ... traffic control
m_root=1000; h_root="${m_root}:"
m_strict=$[${m_root}+100]; h_strict="${m_strict}:"
m_gmb=$[${m_root}+200]; h_gmb="${m_gmb}:"


## Functions
function __start {
  for i in $(seq 0 $[${#TC_IFACES[@]}-1] ); do
    iface=${TC_IFACES[${i}]}
    rate=${TC_RATES[${i}]}

    # Queues
```

```
    h_qos=()

    # ... root
    tc qdisc add dev ${iface} root handle ${h_root} prio bands 2 priomap 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
    h_root_1="${m_root}:1"  # band 0
    h_root_2="${m_root}:2"  # band 1

    # ... strict priority queue # 7 (QoS priority 7)
    tc qdisc add dev ${iface} parent ${h_root_1} handle ${h_strict} sfq perturb 10
    h_qos[7]="${h_root_1}"

    # ... guaranteed minimum bandwidth queues # 6-0 (QoS priority 6-0)
    c_gmb="${m_gmb}:1"
    tc qdisc add dev ${iface} parent ${h_root_2} handle ${h_gmb} htb r2q $[${rate}/1000] default 1${TC_DEFQ}  # default queue # 4 (Qos
priority 4)
    tc class add dev ${iface} parent ${h_gmb} classid ${c_gmb} htb rate ${rate}kbit
    for queue in {6..0}; do
      c_gmb_sub="${m_gmb}:1${queue}"
      tc class add dev ${iface} parent ${c_gmb} classid ${c_gmb_sub} htb rate $[${rate}/100*${q_bw[${queue}]}]kbit ceil ${rate}kbit prio
$[8-${queue}]
      m_gmb_sub=$[${m_gmb}+10+${queue}]; h_gmb_sub="${m_gmb_sub}:"
      tc qdisc add dev ${iface} parent ${c_gmb_sub} handle ${h_gmb_sub} sfq perturb 10
      h_qos[${q_qos[${queue}]}]="${c_gmb_sub}"
    done

    # Traffic classification

    # ... QoS priority 7 (highest)
    #     HA (corosync)
    tc filter add dev ${iface} parent ${h_root} protocol ip pref 1 u32 match ip dport 5404 0xFFFF flowid ${h_qos[7]}
    tc filter add dev ${iface} parent ${h_root} protocol ip pref 2 u32 match ip dport 5405 0xFFFF flowid ${h_qos[7]}

    # ... QoS priority 6 (very high)
    #     VLAN 1 (MGMT)
    [ -e /proc/net/vlan/vlan1 ] && \
      tc filter add dev ${iface} parent ${h_gmb} pref 3 basic match 'meta( vlan eq 1 )' flowid ${h_qos[6]}
    #     Monitoring (nagios)
    tc filter add dev ${iface} parent ${h_gmb} protocol ip pref 4 u32 match ip sport 5666 0xFFFF flowid ${h_qos[6]}

    # ... QoS priority 5 (high)
    #     VLAN 2 (HA)
    [ -e /proc/net/vlan/vlan2 ] && \
      tc filter add dev ${iface} parent ${h_gmb} pref 6 basic match 'meta( vlan eq 2 )' flowid ${h_qos[5]}
    #     System Administration
    tc filter add dev ${iface} parent ${h_gmb} protocol ip pref 9 u32 match ip dst 192.168.4.251/32 flowid ${h_qos[5]}

    # ... QoS priority 4 (medium-high)
    #     VLAN 3 (BACKEND)
```

```
    [ -e /proc/net/vlan/vlan3 ] && \
      tc filter add dev ${iface} parent ${h_gmb} pref 10 basic match 'meta( vlan eq 3 )' flowid ${h_qos[4]}

    # ... QoS priority 3 (medium-low)
    #     VLAN 4 (INTRANET)
    [ -e /proc/net/vlan/vlan4 ] && \
      tc filter add dev ${iface} parent ${h_gmb} pref 11 basic match 'meta( vlan eq 4 )' flowid ${h_qos[3]}

    # ... QoS priority 0 (low, default)
    #     VLAN 5 (DMZ)
    [ -e /proc/net/vlan/vlan5 ] && \
      tc filter add dev ${iface} parent ${h_gmb} pref 13 basic match 'meta( vlan eq 5 )' flowid ${h_qos[0]}

    # ... QoS priority 2 (very low)
    #     VLAN 6 (LAB)
    [ -e /proc/net/vlan/vlan6 ] && \
      tc filter add dev ${iface} parent ${h_gmb} pref 14 basic match 'meta( vlan eq 6 )' flowid ${h_qos[2]}

    # ... QoS priority 1 (lowest)

  done
}

function __stop {
  for i in $(seq 0 $[${#TC_IFACES[@]}-1] ); do
    iface=${TC_IFACES[${i}]}
    tc qdisc del dev ${iface} root 2> /dev/null
  done
}

function __status {
  for i in $(seq 0 $[${#TC_IFACES[@]}-1] ); do
    iface=${TC_IFACES[${i}]}
    echo "========== ${iface} =========="
    echo '---------- filter ----------'
    tc -s filter show dev ${iface} parent ${h_root}
    tc -s filter show dev ${iface} parent ${h_gmb}
    echo '---------- class ----------'
    tc -s class show dev ${iface}
    echo '---------- qdisc ----------'
    tc -s qdisc show dev ${iface}
  done
}


## Actions
. /lib/lsb/init-functions
case "${TC_ACTION}" in
```

```
  start|restart)
    log_daemon_msg "Starting traffic control" "tc"
    __stop
    __start
    log_end_msg 0
    ;;

  stop)
    log_daemon_msg "Stopping traffic control" "tc"
    __stop
    log_end_msg 0
    ;;

  status)
    __status
    ;;

  *)
    echo "ERROR: Invalid action (${TC_ACTION})"
    exit 3
    ;;

esac
exit 0
```

STONITH Plugins

/usr/lib/stonith/plugins/external/ibmbc

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>

# Usage
[ $# -lt 1 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE: ${0##*/} <action> <hostname>

WHERE:
  Actions MUST be one of the following:
    query            - Query the power state for the given host.
    on               - Switch the given host on.
    off              - Switch the given host off.
    reset            - Reset the given host.
    gethosts         - Return the list of hosts configured.
    status           - Exit with return code zero if this device can be reached,
```

```
                        non-zero if this device cannot be reached.
    getconfignames    - Return the list of mandatory environment variables that
                        needs to be configured.
    getinfo-devid     - Return the device class.
    getinfo-devname   - Return the device name.
    getinfo-devdescr  - Return the description of this device.
    getinfo-devurl    - Return a URL pointing to more information on this device.
    getinfo-xml       - Return an XML fragment defining all of the parameters
                        and their descriptions.

REFERENCE:
  See http://www.linux-ha.org/ExternalStonithPlugins
EOF

# Arguments
ARG_ACTION="$1"
ARG_HOSTNAME="$2"

# OIDs
OID_IBMBC_SIG='.1.3.6.1.2.1.1.1.0'
OID_IBMBC_SIG_CHECK='BladeCenter Advanced Management Module'
OID_IBMBC_NAME_PREFIX='.1.3.6.1.4.1.2.3.51.2.22.1.6.1.1.6'
OID_IBMBC_STATE_PREFIX='.1.3.6.1.4.1.2.3.51.2.22.1.6.1.1.4'
OID_IBMBC_POWER_PREFIX='.1.3.6.1.4.1.2.3.51.2.22.1.6.1.1.7'

# Values
VAL_IBMBC_STATE_OFF=0
VAL_IBMBC_STATE_ON=1
VAL_IBMBC_STATE_SOFTOFF=2
VAL_IBMBC_STATE_STANDBY=3
VAL_IBMBC_STATE_HIBERNATE=4

# Set the SNMP configuration directory path
export SNMPCONFPATH="${snmp_conf_path}"

# Useful function
function _devchk() {
  [ -z "${mgmt_address}" ] && echo "ERROR: Missing the device management IP address or hostname (mgmt_address)" >&2 && return 1
  [ -z "${snmp_conf_path}" ] && echo "ERROR: Missing the SNMP configuration directory path (snmp_conf_path)" >&2 && return 1
  [ ! -d "${snmp_conf_path}" ] && echo "ERROR: Invalid SNMP configuration directory path (${snmp_conf_path})" >&2 && return 1
  [ ! -r "${snmp_conf_path}/snmp.conf" ] && echo "ERROR: Missing the SNMP configuration file (${snmp_conf_path}/snmp.conf)" >&2 && return
1
  [ -z "$(which snmpwalk)" ] && echo "ERROR: Missing SNMP binary (snmpwalk)" >&2 && return 1
  [ -z "$(which snmpget)" ] && echo "ERROR: Missing SNMP binary (snmpget)" >&2 && return 1
  [ -z "$(which snmpset)" ] && echo "ERROR: Missing SNMP binary (snmpset)" >&2 && return 1
  sig="$(snmpget -m '' -t 5 -r 15 -Ov -Oq ${mgmt_address} ${OID_IBMBC_SIG} | sed 's|"||g')"
  [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP device signature" >&2 && return 1
  [ "${sig,,}" != "${OID_IBMBC_SIG_CHECK,,}" ] && echo "ERROR: Invalid device signature (${sig})" >&2 && return 1
```

```
  return 0
}

function _gethosts() {
  labels=$(snmpwalk -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMBC_NAME_PREFIX} | sed 's|^"\([^(]*(\)*||;s|\()[^)]*\)*"$||;s| |_|g')
  [ ${PIPESTATUS[0]} -ne 0 ] && echo "ERROR: Failed to retrieve label list" >&2 && return 1
  for label in ${labels}; do
    echo ${label}
  done | uniq
  return 0
}

function _getoids() {
  labels=$(snmpwalk -m '' -t 5 -On -Oq ${mgmt_address} ${OID_IBMBC_NAME_PREFIX} | sed 's|^\([^ ]*\) "\([^(]*(\)*|\1:|;s|\()[^)]*\)*"$||;s|
|_|g')
  [ ${PIPESTATUS[0]} -ne 0 ] && echo "ERROR: Failed to retrieve label list" >&2 && return 1
  found=0
  for label in ${labels}; do
    oid=${label%:*}
    label=${label#*:}
    [ "${label}" != "${ARG_HOSTNAME}" ] && continue
    found=1
    echo ${oid##*.}
  done
  [ ${found} -eq 0 ] && echo "ERROR: No outlet matches the given label (${ARG_HOSTNAME})" >&2 && return 1
  return 0
}

function _query() {
  oids=$(_getoids ${ARG_HOSTNAME}); [ -z "${oids}" ] && return 1
  for oid in ${oids}; do
    state=$(snmpget -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMBC_STATE_PREFIX}.${oid})
    [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP value" >&2 && return 1
    case ${state} in
      ${VAL_IBMBC_STATE_OFF}) echo 'OFF';;
      ${VAL_IBMBC_STATE_ON}|${VAL_IBMBC_STATE_STANDBY}|${VAL_IBMBC_STATE_HIBERNATE}) echo 'ON';;
      *) echo "Unknown power state (${state})";;
    esac
  done
  return 0
}

function _on() {
  oids=$(_getoids ${ARG_HOSTNAME}); [ -z "${oids}" ] && return 1
  for oid in ${oids}; do
    state=$(snmpget -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMBC_STATE_PREFIX}.${oid})
    [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP value" >&2 && return 1
    [ "${state}" == "${VAL_IBMBC_STATE_ON}" ] && continue
```

```
    snmpset -m '' -t 5 ${mgmt_address} ${OID_IBMBC_POWER_PREFIX}.${oid} i ${VAL_IBMBC_STATE_ON} >/dev/null
    [ $? -ne 0 ] && echo "ERROR: Failed to send SNMP command (ON)" >&2 && return 1
    for t in {1..60}; do
      state=$(snmpget -m '' -t 5 -r 1 -Ov -Oq ${mgmt_address} ${OID_IBMBC_STATE_PREFIX}.${oid})
      [ $? -ne 0 ] && echo "WARNING: Failed to retrieve SNMP value" >&2 && continue
      [ "${state}" == "${VAL_IBMBC_STATE_ON}" ] && break
      sleep 1
    done
    [ "${state}" != "${VAL_IBMBC_STATE_ON}" ] && echo "ERROR: Failed to switch the server off (${ARG_HOSTNAME})" >&2 && return 1
  done
  return 0
}

function _off() {
  oids=$(_getoids ${ARG_HOSTNAME}); [ -z "${oids}" ] && return 1
  for oid in ${oids}; do
    state=$(snmpget -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMBC_STATE_PREFIX}.${oid})
    [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP value" >&2 && return 1
    [ "${state}" == "${VAL_IBMBC_STATE_OFF}" ] && continue
    if [ "${no_softoff:-0}" == '0' ]; then
      snmpset -m '' -t 5 ${mgmt_address} ${OID_IBMBC_POWER_PREFIX}.${oid} i ${VAL_IBMBC_STATE_SOFTOFF} >/dev/null
      [ $? -ne 0 ] && echo "ERROR: Failed to send SNMP command (SOFTOFF)" >&2 && return 1
      for t in {1..30}; do
        state=$(snmpget -m '' -t 5 -r 1 -Ov -Oq ${mgmt_address} ${OID_IBMBC_STATE_PREFIX}.${oid})
        [ $? -ne 0 ] && echo "WARNING: Failed to retrieve SNMP value" >&2 && continue
        [ "${state}" == "${VAL_IBMBC_STATE_OFF}" ] && break
        sleep 1
      done
      [ "${state}" == "${VAL_IBMBC_STATE_OFF}" ] && continue
    fi
    snmpset -m '' -t 5 ${mgmt_address} ${OID_IBMBC_POWER_PREFIX}.${oid} i ${VAL_IBMBC_STATE_OFF} >/dev/null
    [ $? -ne 0 ] && echo "ERROR: Failed to send SNMP command (OFF)" >&2 && return 1
    for t in {1..60}; do
      state=$(snmpget -m '' -t 5 -r 1 -Ov -Oq ${mgmt_address} ${OID_IBMBC_STATE_PREFIX}.${oid})
      [ $? -ne 0 ] && echo "WARNING: Failed to retrieve SNMP value" >&2 && continue
      [ "${state}" == "${VAL_IBMBC_STATE_OFF}" ] && break
      sleep 1
    done
    [ "${state}" != "${VAL_IBMBC_STATE_OFF}" ] && echo "ERROR: Failed to switch the server off (${ARG_HOSTNAME})" >&2 && return 1
  done
  return 0
}

# Execute action
case ${ARG_ACTION} in
query)
  _query || exit 1
  exit 0
```

```
  ;;

on)
  _on || exit 1
  exit 0
  ;;

off)
  _off || exit 1
  exit 0
  ;;

reset)
  _off || exit 1
  sleep 1
  _on || exit 1
  exit 0
  ;;

gethosts)
  _gethosts || exit 1
  exit 0
  ;;

status)
  _devchk || exit 1
  exit 0
  ;;

getconfignames)
  echo "mgmt_address"
  echo "snmp_conf_path"
  echo "no_softoff"
  exit 0
  ;;

getinfo-devid)
  echo "IBM BladeCenter STONITH device"
  exit 0
  ;;

getinfo-devname)
  echo "IBM BladeCenter STONITH device (${snmp_conf_path})"
  exit 0
  ;;

getinfo-devdescr)
  echo "IBM BladeCenter's STONITH device via SNMPv3"
```

```
  exit 0
  ;;

getinfo-devurl)
  echo 'http://www.ibm.com/'
  exit 0
  ;;

getinfo-xml)
  cat << EOF
<parameters>

<parameter name="mgmt_address" unique="1" required="1">
<content type="string" />
<shortdesc lang="en">
Management Address (or Hostname)
</shortdesc>
<longdesc lang="en">
The IP address or hostname for the IBM BladeCenter (AMM).
</longdesc>
</parameter>

<parameter name="snmp_conf_path" unique="1" required="1">
<content type="string" />
<shortdesc lang="en">
SNMP Configuration Directory Path
</shortdesc>
<longdesc lang="en">
The path to the directory containing the 'snmp.conf' configuration file to
access the IBM BladeCenter (AMM).
See 'man snmp_config' and 'man snmp.conf' for details.
</longdesc>
</parameter>

<parameter name="no_softoff" unique="1" required="0">
<content type="integer" default="0" />
<shortdesc lang="en">
Do not use SOFTOFF power state to switch host off
</shortdesc>
<longdesc lang="en">
Do not use the SOFTOFF (2) power state when switching host(s) off.
On some systems (eg. IBM BladeServer HX5), power states do not correspond
to documentation: SOFTOFF becomes SOFTRECYCLE and OFF becomes SOFTOFF.
In order to STONITH to work as expected, power 'cycling' SHOULD NOT be used.
</longdesc>
</parameter>

</parameters>
```

```
EOF
  exit 0
  ;;

*)
  echo "ERROR: Invalid action (${ARG_ACTION})" >&2 && exit 1
  exit 1
  ;;

esac
```

/usr/lib/stonith/plugins/external/ibmfx

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>

# Usage
[ $# -lt 1 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE: ${0##*/} <action> <hostname>

WHERE:
  Actions MUST be one of the following:
    query            - Query the power state for the given host.
    on               - Switch the given host on.
    off              - Switch the given host off.
    reset            - Reset the given host.
    gethosts         - Return the list of hosts configured.
    status           - Exit with return code zero if this device can be reached,
                       non-zero if this device cannot be reached.
    getconfignames   - Return the list of mandatory environment variables that
                       needs to be configured.
    getinfo-devid    - Return the device class.
    getinfo-devname  - Return the device name.
    getinfo-devdescr - Return the description of this device.
    getinfo-devurl   - Return a URL pointing to more information on this device.
    getinfo-xml      - Return an XML fragment defining all of the parameters
                       and their descriptions.

REFERENCE:
  See http://www.linux-ha.org/ExternalStonithPlugins
EOF

# Arguments
ARG_ACTION="$1"
ARG_HOSTNAME="$2"

# OIDs
```

```
OID_IBMFX_SIG='.1.3.6.1.2.1.1.1.0'
OID_IBMFX_SIG_CHECK='IBM Flex Chassis Management Module'
OID_IBMFX_NAME_PREFIX='.1.3.6.1.4.1.2.3.51.2.22.1.6.1.1.6'
OID_IBMFX_STATE_PREFIX='.1.3.6.1.4.1.2.3.51.2.22.1.6.1.1.4'
OID_IBMFX_POWER_PREFIX='.1.3.6.1.4.1.2.3.51.2.22.1.6.1.1.7'

# Values
VAL_IBMFX_STATE_OFF=0
VAL_IBMFX_STATE_ON=1
VAL_IBMFX_STATE_SOFTOFF=2
VAL_IBMFX_STATE_STANDBY=3
VAL_IBMFX_STATE_HIBERNATE=4

# Set the SNMP configuration directory path
export SNMPCONFPATH="${snmp_conf_path}"

# Useful function
function _devchk() {
  [ -z "${mgmt_address}" ] && echo "ERROR: Missing the device management IP address or hostname (mgmt_address)" >&2 && return 1
  [ -z "${snmp_conf_path}" ] && echo "ERROR: Missing the SNMP configuration directory path (snmp_conf_path)" >&2 && return 1
  [ ! -d "${snmp_conf_path}" ] && echo "ERROR: Invalid SNMP configuration directory path (${snmp_conf_path})" >&2 && return 1
  [ ! -r "${snmp_conf_path}/snmp.conf" ] && echo "ERROR: Missing the SNMP configuration file (${snmp_conf_path}/snmp.conf)" >&2 && return
1
  [ -z "$(which snmpwalk)" ] && echo "ERROR: Missing SNMP binary (snmpwalk)" >&2 && return 1
  [ -z "$(which snmpget)" ] && echo "ERROR: Missing SNMP binary (snmpget)" >&2 && return 1
  [ -z "$(which snmpset)" ] && echo "ERROR: Missing SNMP binary (snmpset)" >&2 && return 1
  sig="$(snmpget -m '' -t 5 -r 15 -Ov -Oq ${mgmt_address} ${OID_IBMFX_SIG} | sed 's|"||g')"
  [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP device signature" >&2 && return 1
  [ "${sig,,}" != "${OID_IBMFX_SIG_CHECK,,}" ] && echo "ERROR: Invalid device signature (${sig})" >&2 && return 1
  return 0
}

function _gethosts() {
  labels=$(snmpwalk -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMFX_NAME_PREFIX} | sed 's|^"\([^(]*(\)*||;s|\()[^)]*\)*"$||;s| |_|g')
  [ ${PIPESTATUS[0]} -ne 0 ] && echo "ERROR: Failed to retrieve label list" >&2 && return 1
  for label in ${labels}; do
    echo ${label}
  done | uniq
  return 0
}

function _getoids() {
  labels=$(snmpwalk -m '' -t 5 -On -Oq ${mgmt_address} ${OID_IBMFX_NAME_PREFIX} | sed 's|^\([^ ]*\) "\([^(]*(\)*|\1:|;s|\()[^)]*\)*"$||;s|
|_|g')
  [ ${PIPESTATUS[0]} -ne 0 ] && echo "ERROR: Failed to retrieve label list" >&2 && return 1
  found=0
  for label in ${labels}; do
    oid=${label%:*}
```

```
    label=${label#*:}
    [ "${label}" != "${ARG_HOSTNAME}" ] && continue
    found=1
    echo ${oid##*.}
  done
  [ ${found} -eq 0 ] && echo "ERROR: No outlet matches the given label (${ARG_HOSTNAME})" >&2 && return 1
  return 0
}

function _query() {
  oids=$(_getoids ${ARG_HOSTNAME}); [ -z "${oids}" ] && return 1
  for oid in ${oids}; do
    state=$(snmpget -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMFX_STATE_PREFIX}.${oid})
    [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP value" >&2 && return 1
    case ${state} in
      ${VAL_IBMFX_STATE_OFF}) echo 'OFF';;
      ${VAL_IBMFX_STATE_ON}|${VAL_IBMFX_STATE_STANDBY}|${VAL_IBMFX_STATE_HIBERNATE}) echo 'ON';;
      *) echo "Unknown power state (${state})";;
    esac
  done
  return 0
}

function _on() {
  oids=$(_getoids ${ARG_HOSTNAME}); [ -z "${oids}" ] && return 1
  for oid in ${oids}; do
    state=$(snmpget -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMFX_STATE_PREFIX}.${oid})
    [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP value" >&2 && return 1
    [ "${state}" == "${VAL_IBMFX_STATE_ON}" ] && continue
    snmpset -m '' -t 5 ${mgmt_address} ${OID_IBMFX_POWER_PREFIX}.${oid} i ${VAL_IBMFX_STATE_ON} >/dev/null
    [ $? -ne 0 ] && echo "ERROR: Failed to send SNMP command (ON)" >&2 && return 1
    for t in {1..60}; do
      state=$(snmpget -m '' -t 5 -r 1 -Ov -Oq ${mgmt_address} ${OID_IBMFX_STATE_PREFIX}.${oid})
      [ $? -ne 0 ] && echo "WARNING: Failed to retrieve SNMP value" >&2 && continue
      [ "${state}" == "${VAL_IBMFX_STATE_ON}" ] && break
      sleep 1
    done
    [ "${state}" != "${VAL_IBMFX_STATE_ON}" ] && echo "ERROR: Failed to switch the server off (${ARG_HOSTNAME})" >&2 && return 1
  done
  return 0
}

function _off() {
  oids=$(_getoids ${ARG_HOSTNAME}); [ -z "${oids}" ] && return 1
  for oid in ${oids}; do
    state=$(snmpget -m '' -t 5 -Ov -Oq ${mgmt_address} ${OID_IBMFX_STATE_PREFIX}.${oid})
    [ $? -ne 0 ] && echo "ERROR: Failed to retrieve SNMP value" >&2 && return 1
    [ "${state}" == "${VAL_IBMFX_STATE_OFF}" ] && continue
```

```
    if [ "${no_softoff:-0}" == '0' ]; then
      snmpset -m '' -t 5 ${mgmt_address} ${OID_IBMFX_POWER_PREFIX}.${oid} i ${VAL_IBMFX_STATE_SOFTOFF} >/dev/null
      [ $? -ne 0 ] && echo "ERROR: Failed to send SNMP command (SOFTOFF)" >&2 && return 1
      for t in {1..30}; do
        state=$(snmpget -m '' -t 5 -r 1 -Ov -Oq ${mgmt_address} ${OID_IBMFX_STATE_PREFIX}.${oid})
        [ $? -ne 0 ] && echo "WARNING: Failed to retrieve SNMP value" >&2 && continue
        [ "${state}" == "${VAL_IBMFX_STATE_OFF}" ] && break
        sleep 1
      done
      [ "${state}" == "${VAL_IBMFX_STATE_OFF}" ] && continue
    fi
    snmpset -m '' -t 5 ${mgmt_address} ${OID_IBMFX_POWER_PREFIX}.${oid} i ${VAL_IBMFX_STATE_OFF} >/dev/null
    [ $? -ne 0 ] && echo "ERROR: Failed to send SNMP command (OFF)" >&2 && return 1
    for t in {1..60}; do
      state=$(snmpget -m '' -t 5 -r 1 -Ov -Oq ${mgmt_address} ${OID_IBMFX_STATE_PREFIX}.${oid})
      [ $? -ne 0 ] && echo "WARNING: Failed to retrieve SNMP value" >&2 && continue
      [ "${state}" == "${VAL_IBMFX_STATE_OFF}" ] && break
      sleep 1
    done
    [ "${state}" != "${VAL_IBMFX_STATE_OFF}" ] && echo "ERROR: Failed to switch the server off (${ARG_HOSTNAME})" >&2 && return 1
  done
  return 0
}

# Execute action
case ${ARG_ACTION} in
query)
  _query || exit 1
  exit 0
  ;;

on)
  _on || exit 1
  exit 0
  ;;

off)
  _off || exit 1
  exit 0
  ;;

reset)
  _off || exit 1
  sleep 1
  _on || exit 1
  exit 0
  ;;
```

```
gethosts)
  _gethosts || exit 1
  exit 0
  ;;

status)
  _devchk || exit 1
  exit 0
  ;;

getconfignames)
  echo "mgmt_address"
  echo "snmp_conf_path"
  echo "no_softoff"
  exit 0
  ;;

getinfo-devid)
  echo "IBM FlexSystem STONITH device"
  exit 0
  ;;

getinfo-devname)
  echo "IBM FlexSystem STONITH device (${snmp_conf_path})"
  exit 0
  ;;

getinfo-devdescr)
  echo "IBM FlexSystem's STONITH device via SNMPv3"
  exit 0
  ;;

getinfo-devurl)
  echo 'http://www.ibm.com/'
  exit 0
  ;;

getinfo-xml)
  cat << EOF
<parameters>

<parameter name="mgmt_address" unique="1" required="1">
<content type="string" />
<shortdesc lang="en">
Management Address (or Hostname)
</shortdesc>
<longdesc lang="en">
The IP address or hostname for the IBM FlexSystem (CMM).
```

```
</longdesc>
</parameter>

<parameter name="snmp_conf_path" unique="1" required="1">
<content type="string" />
<shortdesc lang="en">
SNMP Configuration Directory Path
</shortdesc>
<longdesc lang="en">
The path to the directory containing the 'snmp.conf' configuration file to
access the IBM FlexSystem (CMM).
See 'man snmp_config' and 'man snmp.conf' for details.
</longdesc>
</parameter>

<parameter name="no_softoff" unique="1" required="0">
<content type="integer" default="0" />
<shortdesc lang="en">
Do not use SOFTOFF power state to switch host off
</shortdesc>
<longdesc lang="en">
Do not use the SOFTOFF (2) power state when switching host(s) off.
On some systems (eg. IBM BladeServer HX5), power states do not correspond
to documentation: SOFTOFF becomes SOFTRECYCLE and OFF becomes SOFTOFF.
In order to STONITH to work as expected, power 'cycling' SHOULD NOT be used.
</longdesc>
</parameter>

</parameters>
EOF
  exit 0
  ;;

*)
  echo "ERROR: Invalid action (${ARG_ACTION})" >&2 && exit 1
  exit 1
  ;;

esac
```

Corosync/Pacemaker Configuration

/etc/corosync/corosync.conf

NOTE: Those are the default settings, as provided by Debian maintainers/package.

```
# Please read the openais.conf.5 manual page

totem {
     version: 2
     token: 3000
     token_retransmits_before_loss_const: 10
     join: 60
     consensus: 3600
     vsftype: none
     max_messages: 20
     clear_node_high_bit: yes
     secauth: off
     threads: 0
     rrp_mode: none
     interface {
          ringnumber: 0
          bindnetaddr: 192.168.2.0
          mcastaddr: 226.94.1.1
          mcastport: 5405
     }
}

amf {
     mode: disabled
}

service {
     ver:       1
     name:      pacemaker
}

aisexec {
        user:   root
        group:  root
}

logging {
        fileline: off
        to_stderr: yes
        to_logfile: no
        to_syslog: yes
     syslog_facility: local0
        debug: off
        timestamp: on
        logger_subsys {
                subsys: AMF
                debug: off
                tags: enter|leave|trace1|trace2|trace3|trace4|trace6
```

```
        }
}
```

crm_config

```
<crm_config>
  <cluster_property_set id="cib-bootstrap-options">
    <nvpair id="cib-bootstrap-options-dc-version" name="dc-version" value="1.1.12-561c4cf"/>
    <nvpair id="cib-bootstrap-options-cluster-infrastructure" name="cluster-infrastructure" value="classic openais (with plugin)"/>
    <nvpair id="cib-bootstrap-options-expected-quorum-votes" name="expected-quorum-votes" value="24"/>
    <nvpair id="cib-bootstrap-options-placement-strategy" name="placement-strategy" value="balanced"/>
    <nvpair id="cib-bootstrap-options-batch-limit" name="batch-limit" value="24"/>
    <nvpair id="cib-bootstrap-options-migration-limit" name="migration-limit" value="5"/>
    <nvpair id="cib-bootstrap-options-last-lrm-refresh" name="last-lrm-refresh" value="1411382456"/>
  </cluster_property_set>
</crm_config>
```

rsc_defaults

```
<rsc_defaults>
  <meta_attributes id="rsc_defaults-options">
    <nvpair id="rsc_defaults-options-resource-stickiness" name="resource-stickiness" value="10000"/>
  </meta_attributes>
</rsc_defaults>
```

LibvirtQemu Resource Agent

/usr/lib/ocf/resource.d/custom/LibvirtQemu

```
#!/bin/bash
#
# License: GNU General Public License (GPL)
#
#   Resource Agent for domains managed by the libvirt API.
#   Requires a running libvirt daemon (libvirtd).
#
#   (c) 2008-2010 Florian Haas, Dejan Muhamedagic,
#                 and Linux-HA contributors
#
# 2014.08.11: Cedric Dufour <cedric.dufour@idiap.ch>
#   Simplified version of 'VirtualDomain' OCF script.
#   (Partially) integrated 'MailTo' OCF script
#
```

```
# Usage: ${0} {start|stop|status|monitor|migrate_to|migrate_from|meta-data|validate-all}
#
##############################################################################
# Initialization:
: ${OCF_FUNCTIONS_DIR=${OCF_ROOT}/lib/heartbeat}
. ${OCF_FUNCTIONS_DIR}/ocf-shellfuncs

# Defaults
OCF_RESKEY_force_stop_default=0
OCF_RESKEY_email_subject='[SYSTEM:HA][VM:%domain_name%]'

: ${OCF_RESKEY_force_stop=${OCF_RESKEY_force_stop_default}}
: ${OCF_RESKEY_email_subject=${OCF_RESKEY_email_subject_default}}
##############################################################################

usage() {
  echo "USAGE: ${0##*/} {start|stop|status|monitor|migrate_to|migrate_from|meta-data|validate-all}"
}

meta_data() {
    cat <<EOF
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="LibvirtQemu">
<version>1.1</version>

<longdesc lang="en">
Resource agent for a libvirt (qemu) virtual domain.
</longdesc>
<shortdesc lang="en">Manages qemu virtual domains through the libvirt virtualization framework</shortdesc>

<parameters>

<parameter name="config" unique="1" required="1">
<longdesc lang="en">
Absolute path to the libvirt (qemu) configuration file (corresponding to the desired virtual domain).
</longdesc>
<shortdesc lang="en">Libvirt (qemu) configuration file</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="force_stop" unique="0" required="0">
<longdesc lang="en">
Always forcefully shut down ("destroy") the domain on stop. The default
behavior is to resort to a forceful shutdown only after a graceful
shutdown attempt has failed. You should only set this to true if
your virtual domain (or your virtualization backend) does not support
graceful shutdown.
```

```
</longdesc>
<shortdesc lang="en">Always force shutdown on stop</shortdesc>
<content type="boolean" default="${OCF_RESKEY_force_stop_default}" />
</parameter>

<parameter name="migration_transport" unique="0" required="0">
<longdesc lang="en">
Transport used to connect to the remote hypervisor while
migrating. Please refer to the libvirt documentation for details on
transports available. If this parameter is omitted, the resource will
use libvirt's default transport to connect to the remote hypervisor.
</longdesc>
<shortdesc lang="en">Remote hypervisor transport</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="migration_network_suffix" unique="0" required="0">
<longdesc lang="en">
Use a dedicated migration network. The migration URI is composed by
adding this parameters value to the end of the node name. If the node
name happens to be an FQDN (as opposed to an unqualified host name),
insert the suffix immediately prior to the first period (.) in the FQDN.

Note: Be sure this composed host name is locally resolveable and the
associated IP is reachable through the favored network.
</longdesc>
<shortdesc lang="en">Migration network host name suffix</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="monitor_scripts" unique="0" required="0">
<longdesc lang="en">
To additionally monitor services within the virtual domain, add this
parameter with a list of scripts to monitor.

Note: when monitor scripts are used, the start and migrate_from operations
will complete only when all monitor scripts have completed successfully.
Be sure to set the timeout of these operations to accommodate this delay.
</longdesc>
<shortdesc lang="en">Space-separated list of monitor scripts</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="email" unique="0" required="0">
<longdesc lang="en">
Space-separated list of operators E-mail addresses (to send status notifications to).
</longdesc>
<shortdesc lang="en">Space-separated E-mail addresses</shortdesc>
```

```
<content type="string" default="" />
</parameter>

<parameter name="email_subject" unique="0" required="0">
<longdesc lang="en">
The subject of the status notification E-mails.

The '%domain_name%' macro shall be replaced with the actual virtual domain name.
</longdesc>
<shortdesc lang="en">E-mail subject</shortdesc>
<content type="string" default="[SYSTEM:HA][VM:%domain_name%]" />
</parameter>

</parameters>

<actions>
<action name="start" timeout="30" />
<action name="stop" timeout="60" />
<action name="status" depth="0" timeout="30" interval="60" />
<action name="monitor" depth="0" timeout="30" interval="60" />
<action name="migrate_from" timeout="60" />
<action name="migrate_to" timeout="60" />
<action name="meta-data" timeout="5" />
<action name="validate-all" timeout="5" />
</actions>
</resource-agent>
EOF
}

# Options to be passed to virsh
VIRSH_OPTIONS="--quiet"

LibvirtQemu_EmailSend() {
  ${MAILCMD} -s "${1}" "${OCF_RESKEY_email}" << EOF
  ${1}
EOF
  return $?
}

LibvirtQemu_Status() {
  local try=0
  local status

  rc=${OCF_ERR_GENERIC}
  status='no state'
  while [ "${status}" == 'no state' ]; do
    try=$(( ${try} + 1 ))
    status="$(virsh ${VIRSH_OPTIONS} domstate ${DOMAIN_NAME} 2>&1)"
```

```
    case "${status,,}" in
      *'domain not found'*|'shut off')
        # shut off: persistent domain is defined, but not started
        # domain not found: domain is not defined and thus not started
        ocf_log debug "Domain '${DOMAIN_NAME}' is currently in state '${status}'."
        rc=${OCF_NOT_RUNNING}
        ;;
      'running'|'paused'|'idle'|'in shutdown'|'blocked')
        # running: domain is currently actively consuming cycles
        # paused: domain is paused (suspended)
        # idle: domain is running but idle
        # in shutdown: domain is being (gracefully) shut down
        # blocked: synonym for idle used by legacy Xen versions
        ocf_log debug "Domain '${DOMAIN_NAME}' is currently in state '${status}'."
        rc=${OCF_SUCCESS}
        ;;
      ''|*'failed to '*'connect to the hypervisor'*|'no state')
        # Empty string may be returned when virsh does not
        # receive a reply from libvirtd.
        # "no state" may occur when the domain is currently
        # being migrated (on the migration target only), or
        # whenever virsh can't reliably obtain the domain
        # state.
        status='no state'
        if [ "${__OCF_ACTION}" == 'stop' ] && [ ${try} -ge 3 ]; then
          # During the stop operation, we want to bail out
          # quickly, so as to be able to force-stop (destroy)
          # the domain if necessary.
          ocfg_log err "Domain '${DOMAIN_NAME}' has no state during stop operation; bailing out."
          return ${OCF_ERR_GENERIC};
        else
          # During all other actions, we just wait and try
          # again, relying on the CRM/LRM to time us out if
          # this takes too long.
          ocf_log info "Domain '${DOMAIN_NAME}' currently has no state; retrying."
          sleep 1
        fi
        ;;
      *)
        # any other output is unexpected.
        ocfg_log err "Domain '${DOMAIN_NAME}' has unknown state ('${status}')!"
        ;;
    esac
  done
  return ${rc}
}

LibvirtQemu_Undefine() {
```

```
  for domain_name in $(virsh ${VIRSH_OPTIONS} list --all --name); do
    if [ "${domain_name}" == "${DOMAIN_NAME}" ]; then
      ocf_log warn "Domain '${DOMAIN_NAME}' is defined as persistent; undefining it (making it transient)"
      virsh ${VIRSH_OPTIONS} undefine ${DOMAIN_NAME} >/dev/null 2>&1
      break
    fi
  done
}

LibvirtQemu_Start() {
  if LibvirtQemu_Status; then
    ocf_log info "Domain '${DOMAIN_NAME}' is already running."
    return ${OCF_SUCCESS}
  fi

  # NOTE: We cannot 'virsh create' a domain that has been previously 'virsh defined'
  LibvirtQemu_Undefine
  virsh ${VIRSH_OPTIONS} create "${OCF_RESKEY_config}"
  rc=$?
  if [ ${rc} -ne 0 ]; then
    ocfg_log err "Failed to start domain '${DOMAIN_NAME}'."
    return ${OCF_ERR_GENERIC}
  fi

  while ! LibvirtQemu_Monitor; do
    sleep 1
  done

  if [ -n "${OCF_RESKEY_email}" ]; then
    LibvirtQemu_EmailSend "${OCF_RESKEY_email_subject//%domain_name%/${DOMAIN_NAME}} $(date +'%Y-%m-%d %H:%M:%S') START on $(uname -n)"
  fi

  return ${OCF_SUCCESS}
}

LibvirtQemu_Stop() {
  local status
  local shutdown_timeout
  local out ex

  LibvirtQemu_Status
  status=$?

  case ${status} in
    ${OCF_SUCCESS})
      if ! ocf_is_true ${OCF_RESKEY_force_stop}; then
        # Issue a graceful shutdown request
        ocf_log info "Issuing graceful shutdown request for domain '${DOMAIN_NAME}'."
```

```
        virsh ${VIRSH_OPTIONS} qemu-monitor-command ${DOMAIN_NAME} --hmp sendkey esc  # For F*%&*% M$ Windaube!...
        virsh ${VIRSH_OPTIONS} shutdown ${DOMAIN_NAME}
        # The "shutdown_timeout" we use here is the operation
        # timeout specified in the CIB, minus 5 seconds
        shutdown_timeout=$(( ${SECONDS} + (${OCF_RESKEY_CRM_meta_timeout}/1000)-5 ))
        # Loop on status until we reach ${shutdown_timeout}
        while [ ${SECONDS} -lt ${shutdown_timeout} ]; do
          LibvirtQemu_Status
          status=$?
          case ${status} in
            ${OCF_NOT_RUNNING})
              # This was a graceful shutdown.
              if [ -n "${OCF_RESKEY_email}" ]; then
                LibvirtQemu_EmailSend "${OCF_RESKEY_email_subject//%domain_name%/${DOMAIN_NAME}} $(date +'%Y-%m-%d %H:%M:%S') STOP
(graceful) on $(uname -n)"
              fi
              return ${OCF_SUCCESS}
              ;;
            ${OCF_SUCCESS})
              # Domain is still running, keep
              # waiting (until shutdown_timeout
              # expires)
              sleep 1
              ;;
            *)
              # Something went wrong. Bail out and
              # resort to forced stop (destroy).
              break;
              ;;
          esac
        done
      fi
      ;;
    ${OCF_NOT_RUNNING})
      ocf_log info "Domain '${DOMAIN_NAME}' already stopped."
      return ${OCF_SUCCESS}
      ;;
  esac
  # OK. Now if the above graceful shutdown hasn't worked, kill
  # off the domain with destroy. If that too does not work,
  # have the LRM time us out.
  ocf_log info "Issuing forced shutdown (destroy) request for domain '${DOMAIN_NAME}'."
  out="$(virsh ${VIRSH_OPTIONS} destroy ${DOMAIN_NAME} 2>&1)"
  ex=$?
  echo "${out}" >&2
  case ${ex}${out,,} in
    *'domain is not running'*|*'domain not found'*)
      : # unexpected path to the intended outcome, all is well
```

```
      ;;
    [!0]*)
      return ${OCF_ERR_GENERIC}
      ;;
    0*)
      while [ ${status} != ${OCF_NOT_RUNNING} ]; do
        LibvirtQemu_Status
        status=$?
      done
      ;;
  esac
  if [ -n "${OCF_RESKEY_email}" ]; then
    LibvirtQemu_EmailSend "${OCF_RESKEY_email_subject//%domain_name%/${DOMAIN_NAME}} $(date +'%Y-%m-%d %H:%M:%S') STOP (forced) on $(uname
-n)"
  fi
  return ${OCF_SUCCESS}
}

LibvirtQemu_Migrate_To() {
  local target_node
  local remoteuri
  local transport_suffix
  local migrateuri
  local migrateport
  local migrate_target

  target_node="${OCF_RESKEY_CRM_meta_migrate_target}"
  if LibvirtQemu_Status; then
    # Find out the remote hypervisor to connect to. That is, turn
    # something like "qemu://foo:9999/system" into
    # "qemu+tcp://bar:9999/system"
    if [ -n "${OCF_RESKEY_migration_transport}" ]; then
      transport_suffix="+${OCF_RESKEY_migration_transport}"
    fi
    # A typical migration URI via a special migration network looks
    # like "tcp://bar-mig:49152". The port would be randomly chosen
    # by libvirt from the range 49152-49215 if omitted, at least since
    # version 0.7.4 ...
    if [ -n "${OCF_RESKEY_migration_network_suffix}" ]; then
      # Hostname might be a FQDN
      migrate_target=$(echo ${target_node} | sed -e "s,^\([^.]\+\),\1${OCF_RESKEY_migration_network_suffix},")
      # For quiet ancient libvirt versions a migration port is needed
      # and the URI must not contain the "//". Newer versions can handle
      # the "bad" URI.
      migrateport=$(( 49152 + $(ocf_maybe_random) % 64 ))
      migrateuri="tcp:${migrate_target}:${migrateport}"
    fi
    remoteuri="qemu${transport_suffix}://${target_node}/system"
```

```
    # OK, we know where to connect to. Now do the actual migration.
    ocf_log info "Migrating domain '${DOMAIN_NAME}' to node '${target_node}' ('${remoteuri}' via '${migrateuri}')."
    virsh ${VIRSH_OPTIONS} migrate --live ${DOMAIN_NAME} ${remoteuri} ${migrateuri}
    rc=$?
    if [ ${rc} -ne 0 ]; then
      ocfg_log err "Migration of domain '${DOMAIN_NAME} to node '${target_node}' ('${remoteuri}' via '${migrateuri}') failed: ${rc}"
      return ${OCF_ERR_GENERIC}
    else
      ocf_log info "Migration of domain '${DOMAIN_NAME}' to node '${target_node}' succeeded."
      if [ -n "${OCF_RESKEY_email}" ]; then
        LibvirtQemu_EmailSend "${OCF_RESKEY_email_subject//%domain_name%/${DOMAIN_NAME}} $(date +'%Y-%m-%d %H:%M:%S') MIGRATE on $(uname
-n) (to ${target_node})"
      fi
      return ${OCF_SUCCESS}
    fi
  else
    ocfg_log err "${DOMAIN_NAME}: migrate_to: Not active locally!"
    return ${OCF_ERR_GENERIC}
  fi
}

LibvirtQemu_Migrate_From() {
  while ! LibvirtQemu_Monitor; do
    sleep 1
  done
  ocf_log info "Migration of domain '${DOMAIN_NAME}' from '${OCF_RESKEY_CRM_meta_migrate_source}' succeeded."
  if [ -n "${OCF_RESKEY_email}" ]; then
    LibvirtQemu_EmailSend "${OCF_RESKEY_email_subject//%domain_name%/${DOMAIN_NAME}} $(date +'%Y-%m-%d %H:%M:%S') MIGRATE on $(uname -n)
(from ${OCF_RESKEY_CRM_meta_migrate_source})"
  fi
  return ${OCF_SUCCESS}
}

LibvirtQemu_Monitor() {
  # First, check the domain status. If that returns anything other
  # than ${OCF_SUCCESS}, something is definitely wrong.
  LibvirtQemu_Status
  rc=$?
  if [ ${rc} -eq ${OCF_SUCCESS} ]; then
    # OK, the generic status check turned out fine. Now, if we
    # have monitor scripts defined, run them one after another.
    for script in ${OCF_RESKEY_monitor_scripts}; do
      script_output="$( ${script} 2>&1)"
      script_rc=$?
      if [ ${script_rc} -ne ${OCF_SUCCESS} ]; then
        # A monitor script returned a non-success exit
        # code. Stop iterating over the list of scripts, log a
```

```
        # warning message, and propagate ${OCF_ERR_GENERIC}.
        ocf_log warn "Monitor script '${script}' for domain '${DOMAIN_NAME}' failed; '${script_output}' [rc=${script_rc}]"
        rc=${OCF_ERR_GENERIC}
        break
      else
        ocf_log debug "Monitor script '${script}' for domain '${DOMAIN_NAME}' succeeded; '${script_output}' [rc=0]"
      fi
    done
  fi
  return ${rc}
}

LibvirtQemu_Validate_All() {
  # Required binaries:
  for binary in virsh grep sed; do
    check_binary ${binary}
  done
  if [ -z "${MAILCMD}" ]; then
    ocfg_log err "MAILCMD variable not set"
    exit ${OCF_ERR_INSTALLED}
  fi
  check_binary "${MAILCMD}"

  if [ -z "${OCF_RESKEY_config}" ]; then
    ocfg_log err "Missing configuration parameter 'config'."
    return ${OCF_ERR_CONFIGURED}
  fi

  # check if we can read the config file (otherwise we're unable to
  # deduce ${DOMAIN_NAME} from it, see below)
  if [ ! -r "${OCF_RESKEY_config}" ]; then
    if ocf_is_probe; then
      ocf_log info "Configuration file '${OCF_RESKEY_config}' not readable during probe."
    else
      ocfg_log err "Configuration file '${OCF_RESKEY_config}' does not exist or is not readable."
      return ${OCF_ERR_INSTALLED}
    fi
  fi
}

if [ $# -ne 1 ]; then
  usage
  exit ${OCF_ERR_ARGS}
fi

case ${1} in
  meta-data)
    meta_data
```

```
    exit ${OCF_SUCCESS}
    ;;
  usage)
    usage
    exit ${OCF_SUCCESS}
    ;;
esac

# Everything except usage and meta-data must pass the validate test
LibvirtQemu_Validate_All || exit $?

# During a probe, it is permissible for the config file to not be
# readable (it might be on shared storage not available during the
# probe). In that case, we're
# unable to get the domain name. Thus, we also can't check whether the
# domain is running. The only thing we can do here is to assume that
# it is not running.
if [ ! -r "${OCF_RESKEY_config}" ]; then
  ocf_is_probe && exit ${OCF_NOT_RUNNING}
  [ "${__OCF_ACTION}" == 'stop' ] && exit ${OCF_SUCCESS}
fi

# Retrieve the domain name from the config file.
DOMAIN_NAME="$(grep '<name>.*</name>' "${OCF_RESKEY_config}" | sed 's/^.*<name>\(.*\)<\/name>.*$/\1/' 2>/dev/null)"
if [ -z "${DOMAIN_NAME}" ]; then
  ocfg_log err "Failed to parse domain name from configuration file ('${OCF_RESKEY_config}')."
  exit ${OCF_ERR_GENERIC}
fi

case ${1} in
  start)
    LibvirtQemu_Start
    ;;
  stop)
    LibvirtQemu_Stop
    ;;
  migrate_to)
    LibvirtQemu_Migrate_To
    ;;
  migrate_from)
    LibvirtQemu_Migrate_From
    ;;
  status)
    LibvirtQemu_Status
    ;;
  monitor)
    LibvirtQemu_Monitor
      ;;
```

```
  validate-all)
    ;;
  *)
    usage
    exit ${OCF_ERR_UNIMPLEMENTED}
    ;;
esac
exit $?
```

Libvirt Sample Configuration

/havc/config/libvirt/template.xml

```
<domain type='kvm'>
  <name>%{VM_FQN}</name>
  <uuid>%{VM_UUID}</uuid>
  <memory>524288</memory>
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  <vcpu>1</vcpu>
<!--
  <cpu match='exact'>
    <model>Nehalem</model>
  </cpu>
-->
  <os>
    <type arch='x86_64' machine='pc-0.12'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='none' error_policy='enospace'/>
      <source file='%{VM_IMAGE}'/>
      <target dev='vda' bus='virtio'/>
```

```
    </disk>
    <interface type='bridge'>
      <mac address='%{MAC_ADDRESS}'/>
      <source bridge='%{NETWORK_BRIDGE}'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='tablet' bus='usb'/>
  </devices>
</domain>
```

Pacemaker Sample Configuration

/havc/config/pacemaker/NETWORK.xml

NOTE: The two IPs used here are those attributed to each of the two core HP E8212zl switches.

```
<resources>
  <clone id="NETWORK">
    <primitive id="NETWORK-ping" class="ocf" provider="pacemaker" type="ping">
      <operations>
        <op id="NETWORK-ping-OP-start" name="start" interval="0" timeout="60s"/>
        <op id="NETWORK-ping-OP-stop" name="stop" interval="0" timeout="30s"/>
        <op id="NETWORK-ping-OP-monitor" name="monitor" interval="60s" timeout="55s"/>
      </operations>
      <instance_attributes id="NETWORK-ping-IA">
        <nvpair id="NETWORK-ping-IA-host_list" name="host_list" value="192.168.3.2 192.168.3.3"/>
        <nvpair id="NETWORK-ping-IA-timeout" name="timeout" value="3"/>
        <nvpair id="NETWORK-ping-IA-attempts" name="attempts" value="3"/>
        <nvpair id="NETWORK-ping-IA-options" name="options" value="-i 3"/>
        <nvpair id="NETWORK-ping-IA-multiplier" name="multiplier" value="500"/>
        <nvpair id="NETWORK-ping-IA-dampen" name="dampen" value="90"/>
      </instance_attributes>
    </primitive>
  </clone>
</resources>
```

/havc/config/pacemaker/STONITH_ibmbc.xml

NOTE: This is a single sample for IBM Blade Center AMM; all IBM (chassis/redundant) AMMs/CMMs shall are configured identically.

```
<resources>
  <clone id="STONITH_ibmbc">
    <meta_attributes id="STONITH_ibmbc-MA">
      <nvpair id="STONITH_ibmbc-MA-globally-unique" name="globally-unique" value="false"/>
    </meta_attributes>
    <primitive id="STONITH_ibmbc-P" class="stonith" type="external/ibmbc">
      <operations>
        <op id="STONITH_ibmbc-P-OP-monitor" name="monitor" interval="21600s" timeout="60s"/>
      </operations>
      <instance_attributes id="STONITH_ibmbc-P-IA">
        <nvpair id="STONITH_ibmbc-P-IA-mgmt_address" name="mgmt_address" value="192.168.1.251"/>
        <nvpair id="STONITH_ibmbc-P-IA-snmp_conf_path" name="snmp_conf_path" value="/havc/config/stonith/ibmbc"/>
        <nvpair id="STONITH_ibmbc-P-IA-no_softoff" name="no_softoff" value="0"/>
      </instance_attributes>
    </primitive>
  </clone>
</resources>
```

/havc/config/pacemaker/resource.template.xml

```
<resources>
  <group id="%{VM_FQN}">
    <primitive id="%{VM_FQN}-LibvirtQemu" class="ocf" provider="custom" type="LibvirtQemu">
      <instance_attributes id="%{VM_FQN}-LibvirtQemu-IA">
        <nvpair id="%{VM_FQN}-LibvirtQemu-IA-config" name="config" value="/havc/config/libvirt/%{VM_FQN}.xml"/>
        <nvpair id="%{VM_FQN}-LibvirtQemu-IA-email" name="email" value="watchdog@example.org"/>
      </instance_attributes>
      <meta_attributes id="%{VM_FQN}-LibvirtQemu-MA">
        <nvpair id="%{VM_FQN}-LibvirtQemu-MA-allow-migrate" name="allow-migrate" value="true"/>
      </meta_attributes>
      <utilization id="%{VM_FQN}-LibvirtQemu-utilization">
        <nvpair id="%{VM_FQN}-LibvirtQemu-utilization-cpu" name="cpu" value="1"/>
        <nvpair id="%{VM_FQN}-LibvirtQemu-utilization-memory" name="memory" value="512"/>
      </utilization>
      <operations>
        <op id="%{VM_FQN}-LibvirtQemu-OP-monitor" name="monitor" timeout="30s" interval="60s"/>
        <op id="%{VM_FQN}-LibvirtQemu-OP-start" name="start" timeout="60s" interval="0"/>
        <op id="%{VM_FQN}-LibvirtQemu-OP-stop" name="stop" timeout="60s" interval="0"/>
        <op id="%{VM_FQN}-LibvirtQemu-OP-migrate-to" name="migrate_to" timeout="60s" interval="0"/>
        <op id="%{VM_FQN}-LibvirtQemu-OP-migrate-from" name="migrate_from" timeout="60s" interval="0"/>
      </operations>
    </primitive>
  </group>
```

```
</resources>
```

/havc/config/pacemaker/constraint.template.xml

```
<constraints>
  <rsc_location id="%{VM_FQN}-connectivity" rsc="%{VM_FQN}">
    <rule id="%{VM_FQN}-connectivity-R" score="-INFINITY" boolean-op="or">
      <expression id="%{VM_FQN}-connectivity-R-E-not_defined" attribute="pingd" operation="not_defined"/>
      <expression id="%{VM_FQN}-connectivity-R-E-lte" attribute="pingd" operation="lte" value="0"/>
    </rule>
  </rsc_location>
  <rsc_location id="%{VM_FQN}-location" rsc="%{VM_FQN}" node="%{PREFERRED_NODE}" score="1000"/>
</constraints>
```

System Administration

/havc/config/pacemaker/ADMIN.xml

```
<resources>
  <group id="ADMIN_havc">
    <primitive id="ADMIN_havc-IPaddr" class="ocf" provider="heartbeat" type="IPaddr2">
      <instance_attributes id="ADMIN_havc-IPaddr-IA">
        <nvpair id="ADMIN_havc-IPaddr-IA-ip" name="ip" value="192.168.3.250"/>
        <nvpair id="ADMIN_havc-IPaddr-IA-cidr_netmask" name="cidr_netmask" value="24"/>
        <nvpair id="ADMIN_havc-IPaddr-IA-nic" name="nic" value="bond0"/>
      </instance_attributes>
    </primitive>
  </group>
</resources>
<constraints>
  <rsc_location id="ADMIN_havc-connectivity" rsc="ADMIN_havc">
    <rule id="ADMIN_havc-connectivity-R" score="-INFINITY" boolean-op="or">
      <expression id="ADMIN_havc-connectivity-R-E-not_defined" attribute="pingd" operation="not_defined"/>
      <expression id="ADMIN_havc-connectivity-R-E-lte" attribute="pingd" operation="lte" value="0"/>
    </rule>
  </rsc_location>
</constraints>
```

/havc/scripts/havc-config-host

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>
```

```
## Usage
[ $# -lt 2 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE:
  ${0##*/} <vm-name> <host-name>

SYNOPSIS:
  Define and save the proper settings for the given host (KVM guest).

WHERE:
  <vm-name>
    Is the Virtual Machine (VM) name, as used by LibVirt (virsh).
    Example: KVMGUEST01

  <host-name>
    Is the host name, as resolvable by the Domain Name Service (DNS).
    Example: kvmguest01.idiap.ch
EOF

# Arguments
VM_NAME="$1"
HOST_NAME="$2"

# Parameters
HOST_CONFIG_DIR='/havc/config/hosts'


## Check (arguments)

# ... virtual machine name
VM_NAME="$(echo "${VM_NAME}" | tr 'abcdefghijklmnopqrstuvwxyz' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"
[ -z "$(echo "${VM_NAME}" | egrep '^[-.A-Z0-9]{3,}$')" ] && echo "ERROR: Invalid virtual machine name (${VM_NAME})" >&2 && exit 1

# ... host name
DNS_LOOKUP="$(host ${HOST_NAME} | egrep ' has address [0-9]{2,3}(\.[0-9]{1,3}){3}$' | head -n 1)"
[ -z "${DNS_LOOKUP}" ] && echo "ERROR: No matching DNS entry for the given hostname (${HOST_NAME})" >&2 && exit 1


## Settings

# ... IP address and FQHN
IP_ADDRESS="$(echo "${DNS_LOOKUP}" | awk '{print $4}')"
[ -z "$(echo "${IP_ADDRESS}" | egrep '^[0-9]{2,3}(\.[0-9]{1,3}){3}$')" ] && echo "ERROR: Invalid IP address (${IP_ADDRESS})" >&2 && exit 1
HOST_FQHN="$(echo "${DNS_LOOKUP}" | awk '{print $1}')"
[ -z "$(echo "${HOST_FQHN}" | egrep '^([-_a-z0-9]{1,}\.){2,}[a-z]{2,4}$')" ] && echo "ERROR: Invalid fully-qualified host name ($
{HOST_FQHN})" >&2 && exit 1
IP_ADDRESS_A=${IP_ADDRESS%%.*}; IP_ADDRESS=${IP_ADDRESS#*.}
IP_ADDRESS_B=${IP_ADDRESS%%.*}; IP_ADDRESS=${IP_ADDRESS#*.}
```

```
IP_ADDRESS_C=${IP_ADDRESS%%.*}; IP_ADDRESS=${IP_ADDRESS#*.}
IP_ADDRESS_D=${IP_ADDRESS%%.*}; IP_ADDRESS=${IP_ADDRESS_A}.${IP_ADDRESS_B}.${IP_ADDRESS_C}.${IP_ADDRESS_D}

# ... network zone
case ${IP_ADDRESS} in

  192.168.4.*) NETWORK_ZONE=intranet; NETWORK_VLAN=4;;
  192.168.5.*) NETWORK_ZONE=dmz; NETWORK_VLAN=5;;
  192.168.6.*) NETWORK_ZONE=lab; NETWORK_VLAN=6;;
  *)  echo "ERROR: Unsupported IP range/address (${IP_ADDRESS})" >&2 && exit 1;;
esac

# ... MAC address and UUID suffix
MAC_ADDRESS="$(printf '02:00:%.2X:%.2X:%.2X:%.2X' ${IP_ADDRESS_A} ${IP_ADDRESS_B} ${IP_ADDRESS_C} ${IP_ADDRESS_D})"
[ -z "$(echo "${MAC_ADDRESS}" | egrep '^[A-F0-9]{2}(:[A-F0-9]{2}){5}$')" ] && echo "ERROR: Invalid MAC address (${MAC_ADDRESS})" >&2 &&
exit 1
UUID_SUFFIX=${MAC_ADDRESS//:/}

# ... IPv6 address
IP_ADDRESS_V6="$(printf '2001:620:7a3:%d::%xff:fe%2x:%x%.2x' ${NETWORK_VLAN} ${IP_ADDRESS_A} ${IP_ADDRESS_B} ${IP_ADDRESS_C} $
{IP_ADDRESS_D})"

# ... virtual machine name
VM_FQN="${VM_NAME}_${NETWORK_ZONE}"

# ... configuration file
HOST_CONFIG_FILE="${HOST_CONFIG_DIR}/${VM_FQN}"
HOST_CONFIG_ALIAS="${HOST_CONFIG_DIR}/${HOST_FQHN}"
if [ -e "${HOST_CONFIG_FILE}" ]; then
  echo "WARNING: The host configuration file already exists (${HOST_CONFIG_FILE})"
  echo 'PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...'
  read
fi


## Feedback
cat << EOF
ABOUT TO CREATE HOST CONFIGURATION:
 - host name: ..... ${HOST_FQHN}
 - network zone: .. ${NETWORK_ZONE}
 - machine name: .. ${VM_FQN}
 - UUID suffix: ... ${UUID_SUFFIX}
 - MAC address: ... ${MAC_ADDRESS}
 - IPv4 address: .. ${IP_ADDRESS}
 - IPv6 address: .. ${IP_ADDRESS_V6}
 - CONFIGURATION: . "${HOST_CONFIG_FILE}"
PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...
EOF
```

```
read


## Commit/save configuration
cat << EOF > "${HOST_CONFIG_FILE}"
HOST_FQHN=${HOST_FQHN}
NETWORK_ZONE=${NETWORK_ZONE}
VM_NAME=${VM_NAME}
UUID_SUFFIX=${UUID_SUFFIX}
MAC_ADDRESS=${MAC_ADDRESS}
IP_ADDRESS=${IP_ADDRESS}
IP_ADDRESS_V6=${IP_ADDRESS_V6}
EOF
ln -s "${HOST_CONFIG_FILE##*/}" "${HOST_CONFIG_ALIAS}"


## DONE
echo
cat << EOF
HOST CONFIGURATION CREATED!
You can now:
  # Create the corresponding KVM/LibVirt configuration
    > havc-config-libvirt ${VM_FQN} <disk-source>
  # Create the corresponding HA/Pacemaker configuration
    > havc-config-pacemaker ${VM_FQN} <preferred-node>
EOF
echo
```

/havc/scripts/havc-config-libvirt

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Usage
[ $# -lt 2 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE:
  ${0##*/} <vm-fqn|host-fqhn> <disk-source> [<libvirt-template>=virtual-server-linux]

SYNOPSIS:
  Configure the KVM/LibVirt stack for the given host (KVM guest).

WHERE:
  <vm-fqn>
    Is the fully qualified virtual machine name (as configured by the 'havc-config-host' script).
    Example: KVMGUEST01_intranet
```

```
  <host-fqhn>
    Is the fully qualified host name (as configured by the 'havc-config-host' script).
    Example: kvmguest01.idiap.ch

  <image-source>
    Is the image source (path) to use as the actual image for
    the Virtual Machine.
    Example: /havc/storage/filer/vm/intranet/newguest.raw

  <libvirt-template>
    Is the XML filename used as template for defining the
    virtual machine (using LibVirt stanza).
    Default: virtual-server-linux
EOF

# Arguments
HOST_CONFIG_NAME="$1"
VM_IMAGE_SOURCE="$2"
[ $# -ge 3 ] && LIBVIRT_TEMPLATE="$3" || LIBVIRT_TEMPLATE='virtual-server-linux'

# Parameters
HOST_CONFIG_DIR='/havc/config/hosts'
LIBVIRT_CONFIG_DIR='/havc/config/libvirt'
LIBVIRT_TEMPLATE_DIR='/havc/config/libvirt/TEMPLATES'
VM_STORAGE_DIRS='/dev/ /havc/storage/filer/vm/intranet/ /havc/storage/filer/vm/dmz/ /havc/storage/filer/vm/lab/'


## Check (dependencies and arguments)

# ... host configuration
HOST_CONFIG_FILE="${HOST_CONFIG_DIR}/${HOST_CONFIG_NAME}"
[ ! -r "${HOST_CONFIG_FILE}" ] && echo "ERROR: Missing/unreadable host configuration file (${HOST_CONFIG_FILE})" >&2 && exit 1

# ... installation source
[ ! -e "${VM_IMAGE_SOURCE}" ] && echo "ERROR: Missing/invalid VM image source (${VM_IMAGE_SOURCE})" >&2 && exit 1
VM_IMAGE_VALID=''
for vm_storage_dir in ${VM_STORAGE_DIRS}; do
  if [ "${VM_IMAGE_SOURCE:0:${#vm_storage_dir}}" == "${vm_storage_dir}" ]; then
    VM_IMAGE_VALID='yes'
    break
  fi
done
[ "${VM_IMAGE_VALID}" != 'yes' ] && echo "ERROR: VM image source MUST be on a valid HAVC storage location (${VM_IMAGE_SOURCE})" >&2 &&
exit 1


## Load configuration
HOST_FQHN=; IP_ADDRESS=; MAC_ADDRESS=; NETWORK_ZONE=; VM_NAME=; UUID_SUFFIX=;
```

```
source "${HOST_CONFIG_FILE}"


## Create/check (configuration)

# ... virtual machine name
VM_FQN="${VM_NAME}_${NETWORK_ZONE}"

# ... network zone
case "${NETWORK_ZONE}" in
  'intranet') NETWORK_BRIDGE='br0' ;;
  'dmz') NETWORK_BRIDGE='br1' ;;
  'lab') NETWORK_BRIDGE='br2' ;;
  *) echo "ERROR: Invalid network zone (${NETWORK_ZONE})" >&2 && exit 1
esac

# ... disk image
if [ "${VM_IMAGE_SOURCE:0:5}" == '/dev/' ]; then
  VM_IMAGE="${VM_IMAGE_SOURCE}"
else
  VM_IMAGE="${VM_IMAGE_SOURCE%/*}/${VM_FQN}.raw"
  if [ -e "${VM_IMAGE}" ]; then
    echo "WARNING: The virtual machine storage image already exists (${VM_IMAGE})"
    echo 'PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...'
    read
  fi
fi

# ... configuration files
LIBVIRT_TEMPLATE_FILE="${LIBVIRT_TEMPLATE_DIR}/${LIBVIRT_TEMPLATE}.xml"
[ ! -r "${LIBVIRT_TEMPLATE_FILE}" ] && echo "ERROR: Missing/unreadable KVM/LibVirt template file (${LIBVIRT_TEMPLATE_FILE})" >&2 && exit 1
LIBVIRT_CONFIG_FILE="${LIBVIRT_CONFIG_DIR}/${VM_FQN}.xml"
if [ -e "${LIBVIRT_CONFIG_FILE}" ]; then
  echo "WARNING: The KVM/LibVirt configuration file already exists (${LIBVIRT_CONFIG_FILE})"
  echo 'PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...'
  read
fi


## Feedback
cat << EOF
ABOUT TO CREATE KVM/LIBVIRT CONFIGURATION:
 - host name: ...... ${HOST_FQHN}
 - IP address: ..... ${IP_ADDRESS}
 - MAC address: .... ${MAC_ADDRESS}
 - network zone: ... ${NETWORK_ZONE}
 - machine name: ... ${VM_FQN}
 - UUID suffix: .... ${UUID_SUFFIX}
```

```
 - network bridge: . ${NETWORK_BRIDGE}
 - image source: ... "${VM_IMAGE_SOURCE}"
 - actual image: ... "${VM_IMAGE}"
 - template: ....... "${LIBVIRT_TEMPLATE_FILE}"
 - CONFIGURATION: .. "${LIBVIRT_CONFIG_FILE}"
PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...
EOF
read


## Commit/save configuration

# Create image
if [ "${VM_IMAGE}" != "${VM_IMAGE_SOURCE}" ]; then
  echo "INFO: Importing the virtual machine image"
  mv -v "${VM_IMAGE_SOURCE}" "${VM_IMAGE}"
  chmod -v o= "${VM_IMAGE}"
fi

# Save configuration and (re-)create virtual machine
sed "s/%{HOST_NAME}/${HOST_FQHN}/g;s/%{IP_ADDRESS}/${IP_ADDRESS}/g;s/%{MAC_ADDRESS}/${MAC_ADDRESS}/g;s/%{NETWORK_ZONE}/$
{NETWORK_ZONE}/g;s/%{VM_FQN}/${VM_FQN}/g;s/%{UUID_SUFFIX}/${UUID_SUFFIX}/g;s/%{NETWORK_BRIDGE}/${NETWORK_BRIDGE}/g;s:%{VM_IMAGE}:$
{VM_IMAGE}:g" "${LIBVIRT_TEMPLATE_FILE}" > "${LIBVIRT_CONFIG_FILE}"

## DONE
echo
cat << EOF
KVM/LIBVIRT CONFIGURATION CREATED!
You can now:
  # Create the corresponding HA/Pacemaker configuration
    > havc-config-pacemaker ${VM_FQN} <preferred-node>
  # Manually start the virtual machine (ARE YOU SURE?)
    > virsh create "${LIBVIRT_CONFIG_FILE}"
EOF
echo
```

/havc/scripts/havc-config-pacemaker

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Usage
[ $# -lt 2 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE:
  ${0##*/} <vm-fqn|host-fqhn> <preferred-node> [<resource-template>=virtual-server] [<contsraint-template>=virtual-server]
```

```
SYNOPSIS:
  Configure the HA/Pacemaker stack for the given host (KVM guest).

WHERE:
  <vm-fqn>
    Is the fully qualified virtual machine name (as configured by the 'havc-config-host' script).
    Example: KVMGUEST01_intranet

  <host-fqhn>
    Is the fully qualified host name (as configured by the 'havc-config-host' script).
    Example: kvmguest01.idiap.ch

  <preferred-node>
    Is the preferred node for the resource location.
    Example: hvmhost01

  <resource-template>
     Is the XML filename used as template for defining the
     HA resource.
     Default: virtual-server

  <constraint-template>
     Is the XML filename used as template for defining the
     corresponding HA constraint(s).
     Default: connectivity
EOF

# Arguments
HOST_CONFIG_NAME="$1"
PREFERRED_NODE="${2}"
[ $# -ge 3 ] && RESOURCE_TEMPLATE="$3" || RESOURCE_TEMPLATE='virtual-server'
[ $# -ge 4 ] && CONSTRAINT_TEMPLATE="$4" || CONSTRAINT_TEMPLATE='virtual-server'

# Parameters
HOST_CONFIG_DIR='/havc/config/hosts'
PACEMAKER_CONFIG_DIR='/havc/config/pacemaker'
PACEMAKER_TEMPLATE_DIR='/havc/config/pacemaker/TEMPLATES'


## Check (dependencies and arguments)

# ... host configuration
HOST_CONFIG_FILE="${HOST_CONFIG_DIR}/${HOST_CONFIG_NAME}"
[ ! -r "${HOST_CONFIG_FILE}" ] && echo "ERROR: Missing/unreadable host configuration file (${HOST_CONFIG_FILE})" >&2 && exit 1


## Load configuration
HOST_FQHN=; IP_ADDRESS=; MAC_ADDRESS=; NETWORK_ZONE=; VM_NAME=; UUID_SUFFIX=;
```

```
source "${HOST_CONFIG_FILE}"


## Check/create (configuration)

# ... network zone
case "${NETWORK_ZONE}" in
  'intranet') NETWORK_BRIDGE='br0' ;;
  'dmz') NETWORK_BRIDGE='br1' ;;
  'lab') NETWORK_BRIDGE='br2' ;;
  *) echo "ERROR: Invalid network zone (${NETWORK_ZONE})" >&2 && exit 1
esac

# ... virtual machine name
VM_FQN="${VM_NAME}_${NETWORK_ZONE}"

# ... IP address
[ -z "$(echo "${IP_ADDRESS}" | egrep '^[0-9]{2,3}(\.[0-9]{1,3}){3}$')" ] && echo "ERROR: Invalid IP address (${IP_ADDRESS})" >&2 && exit 1

# ... resource configuration files
RESOURCE_TEMPLATE_FILE="${PACEMAKER_TEMPLATE_DIR}/resource.${RESOURCE_TEMPLATE}.xml"
[ ! -r "${RESOURCE_TEMPLATE_FILE}" ] && echo "ERROR: Missing/unreadable HA/Pacemaker resource template file (${RESOURCE_TEMPLATE_FILE})"
>&2 && exit 1
RESOURCE_CONFIG_FILE="${PACEMAKER_CONFIG_DIR}/${VM_FQN}.resource.xml"
if [ -e "${RESOURCE_CONFIG_FILE}" ]; then
  echo "WARNING: The HA/Pacemaker resource configuration file already exists (${RESOURCE_CONFIG_FILE})"
  echo 'PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...'
  read
fi

# ... constraint configuration files
CONSTRAINT_TEMPLATE_FILE="${PACEMAKER_TEMPLATE_DIR}/constraint.${CONSTRAINT_TEMPLATE}.xml"
[ ! -r "${CONSTRAINT_TEMPLATE_FILE}" ] && echo "ERROR: Missing/unreadable HA/Pacemaker constraint template file ($
{CONSTRAINT_TEMPLATE_FILE})" >&2 && exit 1
CONSTRAINT_CONFIG_FILE="${PACEMAKER_CONFIG_DIR}/${VM_FQN}.constraint.xml"
if [ -e "${CONSTRAINT_CONFIG_FILE}" ]; then
  echo "WARNING: The HA/Pacemaker constraint configuration file already exists (${CONSTRAINT_CONFIG_FILE})"
  echo 'PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...'
  read
fi


## Feedback
cat << EOF
ABOUT TO CREATE HA/PACEMAKER CONFIGURATION:
 - host name: ................ ${HOST_FQHN}
 - IP address: ............... ${IP_ADDRESS}
 - MAC address: .............. ${MAC_ADDRESS}
```

```
 - network zone: ............. ${NETWORK_ZONE}
 - machine name: ............. ${VM_FQN}
 - UUID suffix: .............. ${UUID_SUFFIX}
 - network bridge: ........... ${NETWORK_BRIDGE}
 - preferred node(s): ........ ${PREFERRED_NODE}${PREFERRED_NODE_2:+,}${PREFERRED_NODE_2}
 - resource template: ........ "${RESOURCE_TEMPLATE_FILE}"
 - RESOURCE CONFIGURATION: ... "${RESOURCE_CONFIG_FILE}"
 - constraint template: ...... "${CONSTRAINT_TEMPLATE_FILE}"
 - CONSTRAINT CONFIGURATION: . "${CONSTRAINT_CONFIG_FILE}"
PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...
EOF
read


## Commit/save configuration
sed_expr="s/%{HOST_NAME}/${HOST_FQHN}/g;s/%{IP_ADDRESS}/${IP_ADDRESS}/g;s/%{MAC_ADDRESS}/${MAC_ADDRESS}/g;s/%{NETWORK_ZONE}/$
{NETWORK_ZONE}/g;s/%{VM_FQN}/${VM_FQN}/g;s/%{UUID_SUFFIX}/${UUID_SUFFIX}/g;s/%{NETWORK_BRIDGE}/${NETWORK_BRIDGE}/g;s/%{PREFERRED_NODE}/$
{PREFERRED_NODE}/g"
sed_expr="${sed_expr};/%{PREFERRED_NODE_2}/d"
sed "${sed_expr}" "${RESOURCE_TEMPLATE_FILE}" > "${RESOURCE_CONFIG_FILE}"
sed "${sed_expr}" "${CONSTRAINT_TEMPLATE_FILE}" > "${CONSTRAINT_CONFIG_FILE}"
unset sed_expr


## DONE
echo
cat << EOF
HA/PACEMAKER CONFIGURATION CREATED!
You can now:
  # Configure allocated hardware resources
    > havc-config-hardware ${VM_FQN} <RAM-megabytes> [<CPU-cores>=1]
  # Enable the HA/Pacemaker resource
    > havc-enable ${VM_FQN}
EOF
echo
```

/havc/scripts/havc-config-hardware

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Usage
[ $# -lt 2 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE:
  ${0##*/} <vm-fqn|host-fqhn> <RAM-megabytes> [<CPU-cores>=1]
```

```
SYNOPSIS:
  Configure the CPU/RAM hardware resources for the given host (KVM guest).

WHERE:
  <vm-fqn>
    Is the fully qualified virtual machine name (as configured by the 'havc-config-host' script).
    Example: KVMGUEST01_intranet

  <host-fqhn>
    Is the fully qualified host name (as configured by the 'havc-config-host' script).
    Example: kvmguest01.idiap.ch

  <RAM-megabytes>
     Is the quantity of RAM, in megabytes (MiB).

  <CPU-cores>
    Is the number of vCPU(s).
    WARNING: Do NOT over-commit KVM guests with vCPUs>=2 !
EOF

# Arguments
HOST_CONFIG_NAME="$1"
HARDWARE_RAM="$2"
[ $# -ge 3 ] && HARDWARE_CPU="$3" || HARDWARE_CPU=1

# Parameters
HOST_CONFIG_DIR='/havc/config/hosts'
LIBVIRT_CONFIG_DIR='/havc/config/libvirt'
PACEMAKER_CONFIG_DIR='/havc/config/pacemaker'


## Check (dependencies and arguments)

# ... host configuration
HOST_CONFIG_FILE="${HOST_CONFIG_DIR}/${HOST_CONFIG_NAME}"
[ ! -r "${HOST_CONFIG_FILE}" ] && echo "ERROR: Missing/unreadable host configuration file (${HOST_CONFIG_FILE})" >&2 && exit 1


## Load configuration
HOST_FQHN=; IP_ADDRESS=; MAC_ADDRESS=; NETWORK_ZONE=; VM_NAME=; UUID_SUFFIX=;
source "${HOST_CONFIG_FILE}"

# ... virtual machine name
VM_FQN="${VM_NAME}_${NETWORK_ZONE}"

# ... libvirt configuration
LIBVIRT_CONFIG_FILE="${LIBVIRT_CONFIG_DIR}/${VM_FQN}.xml"
[ ! -r "${LIBVIRT_CONFIG_FILE}" ] && echo "ERROR: Missing/unreadable Libvirt configuration file (${LIBVIRT_CONFIG_FILE})" >&2 && exit 1
```

```
fgrep -q '<memory>' "${LIBVIRT_CONFIG_FILE}"
[ $? -ne 0 ] && echo "ERROR: Missing 'memory' node in Libvirt configuration file (${LIBVIRT_CONFIG_FILE})" >&2 && exit 1
fgrep -q '<vcpu>' "${LIBVIRT_CONFIG_FILE}"
[ $? -ne 0 ] && echo "ERROR: Missing 'vcpu' node in Libvirt configuration file (${LIBVIRT_CONFIG_FILE})" >&2 && exit 1

# ... pacemaker configuration
PACEMAKER_CONFIG_FILE="${PACEMAKER_CONFIG_DIR}/${VM_FQN}.resource.xml"
[ ! -r "${PACEMAKER_CONFIG_FILE}" ] && echo "ERROR: Missing/unreadable HA/Pacemaker configuration file (${PACEMAKER_CONFIG_FILE})" >&2 &&
exit 1
fgrep -q 'name="memory"' "${PACEMAKER_CONFIG_FILE}"
[ $? -ne 0 ] && echo "ERROR: Missing 'memory' node in HA/Pacemaker configuration file (${PACEMAKER_CONFIG_FILE})" >&2 && exit 1
fgrep -q 'name="cpu"' "${PACEMAKER_CONFIG_FILE}"
[ $? -ne 0 ] && echo "ERROR: Missing 'cpu' node in HA/Pacemaker configuration file (${PACEMAKER_CONFIG_FILE})" >&2 && exit 1


## Feedback
cat << EOF
ABOUT TO UPDATE LIBVIRT/PACEMAKER CONFIGURATION:
 - host name: ................ ${HOST_FQHN}
 - machine name: ............. ${VM_FQN}
 - RAM(MiB):.................. ${HARDWARE_RAM}
 - vCPU(s): .................. ${HARDWARE_CPU}
 - LIBVIRT CONFIGURATION: .... "${LIBVIRT_CONFIG_FILE}"
 - PACEMAKER CONFIGURATION: .. "${PACEMAKER_CONFIG_FILE}"
PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...
EOF
read


## Update configuration

# ... libvirt
sed -i "s|<memory>.*$|<memory>$(( ${HARDWARE_RAM}*1024 ))</memory>|;s|<vcpu>.*$|<vcpu>${HARDWARE_CPU}</vcpu>|" "${LIBVIRT_CONFIG_FILE}"
xmllint --debug "${LIBVIRT_CONFIG_FILE}" >/dev/null || exit 1

# ... pacemaker
sed -i "s|name=\"memory\"\(.*\)value=\"[^\"]*\"|name=\"memory\"\1value=\"${HARDWARE_RAM}\"|;s|name=\"cpu\"\(.*\)value=\"[^\"]*\"|
name=\"cpu\"\1value=\"${HARDWARE_CPU}\"|" "${PACEMAKER_CONFIG_FILE}"
xmllint --debug "${PACEMAKER_CONFIG_FILE}" >/dev/null || exit 1


## DONE
echo
cat << EOF
LIBVIRT/PACEMAKER CONFIGURATION UPDATED!
You MUST now:
  # Disable the virtual machine
    > havc-disable ${VM_FQN}
```

```
  # Re-enable the virtual machine
    > havc-enable ${VM_FQN}
EOF
echo
```

/havc/scripts/havc-sync

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Usage
[ "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE: ${0##*/} [--cleanup] [--I-KNOW-WHAT-I-AM-DOING]
EOF

# Arguments
MY_CLEANUP=
MY_CONFIRM=
while [ -n "${1}" ]; do
  case "${1}" in
    --cleanup) MY_CLEANUP='yes';;
    --I-KNOW-WHAT-I-AM-DOING) MY_CONFIRM='yes';;
  esac
  shift
done

# Parameters
MY_RSYNC_OPTIONS=
if [ "${MY_CLEANUP}" == 'yes' ]; then
  MY_RSYNC_OPTIONS='--verbose --delete --force'
  [ "${MY_CONFIRM}" != 'yes' ] && MY_RSYNC_OPTIONS="${MY_RSYNC_OPTIONS} --dry-run"
fi


## Synchronize HAVC resources
echo 'INFO: Synchronizing HAVC resources (config, scripts, etc.)'
eval "rsync -a ${MY_RSYNC_OPTIONS} --exclude=TEMPLATES --exclude=ARCHIVES /havc/storage/filer/hypervisor/config/ /havc/config/"
eval "rsync -a ${MY_RSYNC_OPTIONS} --exclude=ARCHIVES /havc/storage/filer/hypervisor/scripts/ /havc/scripts/"
```

/havc/scripts/havc-enable

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


```

```
## Usage
[ $# -lt 1 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE:
  ${0##*/} <vm-fqn>

SYNOPSIS:
  Enable the given host (KVM guest) in the HA/Pacemaker stack.

WHERE:
  <vm-fqn>
    Is the fully qualified virtual machine name (as configured by the 'havc-config-host' script).
    Example: KVMGUEST01_intranet
EOF

# Arguments
HOST_CONFIG_NAME="$1"

# Parameters
HOST_CONFIG_DIR='/havc/config/hosts'
PACEMAKER_CONFIG_DIR='/havc/config/pacemaker'


## Check (dependencies and arguments)

# ... dependencies
[ -z "$(which cibadmin)" ] && echo 'ERROR: Missing required depencency (cibadmin)' >&2 && exit 1

# ... host configuration
HOST_CONFIG_FILE="${HOST_CONFIG_DIR}/${HOST_CONFIG_NAME}"
[ ! -r "${HOST_CONFIG_FILE}" ] && echo "ERROR: Missing/unreadable host configuration file (${HOST_CONFIG_FILE})" >&2 && exit 1


## Load configuration
HOST_FQHN=; IP_ADDRESS=; MAC_ADDRESS=; NETWORK_ZONE=; VM_NAME=; UUID_SUFFIX=;
source "${HOST_CONFIG_FILE}"


## Check/create (configuration)

# ... virtual machine name
VM_FQN="${VM_NAME}_${NETWORK_ZONE}"

# ... resource configuration files
RESOURCE_CONFIG_FILE="${PACEMAKER_CONFIG_DIR}/${VM_FQN}.resource.xml"
if [ ! -r "${RESOURCE_CONFIG_FILE}" ]; then
  echo "ERROR: The HA/Pacemaker resource configuration file is missing/unreadable (${RESOURCE_CONFIG_FILE})" >&2
  exit 1
fi
```

```
# ... constraint configuration files
CONSTRAINT_CONFIG_FILE="${PACEMAKER_CONFIG_DIR}/${VM_FQN}.constraint.xml"
if [ ! -r "${CONSTRAINT_CONFIG_FILE}" ]; then
  echo "WARNING: The HA/Pacemaker constraint configuration file is missing unreadable (${CONSTRAINT_CONFIG_FILE})"
  echo 'PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...'
  read
fi


## Feedback
cat << EOF
ABOUT TO UPDATE HA/PACEMAKER CONFIGURATION:
 - host name: ................ ${HOST_FQHN}
 - IP address: ............... ${IP_ADDRESS}
 - MAC address: .............. ${MAC_ADDRESS}
 - network zone: ............. ${NETWORK_ZONE}
 - machine name: ............. ${VM_FQN}
 - UUID suffix: .............. ${UUID_SUFFIX}
 - RESOURCE CONFIGURATION: ... "${RESOURCE_CONFIG_FILE}"
 - CONSTRAINT CONFIGURATION: . "${CONSTRAINT_CONFIG_FILE}"
PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...
EOF
read


## Commit
cibadmin -o resources -M -c -x "${RESOURCE_CONFIG_FILE}"
[ -r "${CONSTRAINT_CONFIG_FILE}" ] && cibadmin -o constraints -M -c -x "${CONSTRAINT_CONFIG_FILE}"

## DONE
echo
cat << EOF
HA/PACEMAKER CONFIGURATION UPDATED!
You can now:
  # Display the HA/Pacemaker status
    > ha-status ${VM_FQN}
EOF
echo
```

/havc/scripts/havc-disable

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Usage
```

```
[ $# -lt 1 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE:
  ${0##*/} <vm-fqn>

SYNOPSIS:
  Disable the given host (KVM guest) in the HA/Pacemaker stack.

WHERE:
  <vm-fqn>
    Is the fully qualified virtual machine name (as configured by the 'havc-config-host' script).
    Example: KVMGUEST01_intranet
EOF

# Arguments
VM_FQN="$1"


## Check (dependencies and arguments)

# ... dependencies
[ -z "$(which cibadmin)" ] && echo 'ERROR: Missing required depencency (cibadmin)' >&2 && exit 1
[ -z "$(which crm)" ] && echo 'ERROR: Missing required depencency (crm)' >&2 && exit 1


## Check/retrieve (configuration)

# ... resource ID
HA_RESOURCE_XML="$(cibadmin -o resources -Q | fgrep "id=\"${VM_FQN}\"")"
[ -z "${HA_RESOURCE_XML}" ] && echo "ERROR: Missing/unmatchable virtual machine name (${VM_FQN})" >&2 && exit 1
HA_RESOURCE_NAME="$(echo "${HA_RESOURCE_XML}" | sed 's/.*id="\([^"]*\)".*/\1/')"

# ... constraints IDs
HA_CONSTRAINTS_XML="$(cibadmin -o constraints -Q | fgrep "rsc=\"${VM_FQN}\"")"
HA_CONSTRAINTS_NAME="$(echo "${HA_CONSTRAINTS_XML}" | sed 's/.*id="\([^"]*\)".*/\1/' | tr '\n' ',')"


## Feedback
cat << EOF
ABOUT TO UPDATE HA/PACEMAKER CONFIGURATION:
 - machine name: ............. ${VM_FQN}
 - RESOURCE ID: .............. ${HA_RESOURCE_NAME}
 - CONSTRAINT(S) ID(S): ...... ${HA_CONSTRAINTS_NAME%%,}
PRESS <ENTER> TO CONTINUE, <CTRL+C> TO ABORT...
EOF
read


## Traps
```

```
HA_INTERRUPTED=0
trap 'HA_INTERRUPTED=1' SIGINT


## Commit
crm resource stop ${HA_RESOURCE_NAME}
echo 'INFO: Waiting for resource to stop...'
echo '1-------10--------20--------30--------40--------50--------60'
HA_COUNT_START="${SECONDS}"
HA_COUNT_DOTS=0
while true; do
  [ ${HA_INTERRUPTED} -ne 0 ] && break
  [ -n "$(crm resource status ${HA_RESOURCE_NAME} 2>&1 | fgrep -i 'not running')" ] && break
  HA_COUNT_ELAPSED=$(( ${SECONDS} - ${HA_COUNT_START} ))
  while [ ${HA_COUNT_DOTS} -lt ${HA_COUNT_ELAPSED} ]; do
    HA_COUNT_DOTS=$(( ${HA_COUNT_DOTS} + 1 ))
    echo -n '.'
    [ $(( ${HA_COUNT_DOTS} % 60 )) -eq 0 ] && echo
  done
  sleep 1
done
echo
if [ ${HA_INTERRUPTED} -ne 0 ]; then
  echo 'WARNING: Interrupted while waiting for resource to stop!'
  echo 'WARNING: Resource will eventually be stopped but will NOT be disabled!'
  exit 1
fi
echo 'INFO: Resource successfully stopped'
IFS=$'\n'
for xml in ${HA_CONSTRAINTS_XML}; do cibadmin -o constraints -D -X "${xml}"; done
for xml in ${HA_RESOURCE_XML}; do cibadmin -o resources -D -X "${xml}"; done


## DONE
exit 0
```

/havc/scripts/havc-health

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>

# Track errors/warnings
HA_WARNINGS=0
HA_ERRORS=0

# Check mountpoints
for d in /havc/storage/gx05/{hypervisor,vm/intranet,vm/dmz,vm/lab}; do
```

```
  echo "INFO: Testing mount '${d}'"
  [ ! -d "${d}" ] && echo "ERROR: Invalid/missing mountpoint (${d})" && HA_ERRORS=$(( ${HA_ERRORS}+1 )) && continue
  [ -z "$(awk "{if(\$2==\"${d}\")print \$2}" /proc/mounts)" ] && echo "ERROR: Missing mount (${d})" && HA_ERRORS=$(( ${HA_ERRORS}+1 )) &&
continue
done

# Check peers
HA_HOST="$(hostname -s)"
if [ "${HA_HOST:0:5}" == 'janus' ]; then
  HA_HOSTS="$(echo janus0{1..6})"
else
  HA_HOSTS="$(echo bc{1,2}{hs22a0{1..3},hx5a0{4..7}} fx{1,2}x240a0{1..4})"
fi
n=0; for p in ${HA_HOSTS}; do
  echo "INFO: Testing peer '${p}'"
  ping -c 1 -w 1 ${p} >/dev/null
  [ $? -ne 0 ] && echo "WARNING: Peer does not respond to ping (${p})" && HA_WARNINGS=$(( ${HA_WARNINGS}+1 )) && continue
  virsh --quiet --connect qemu://${p}/system uri >/dev/null
  [ $? -ne 0 ] && echo "WARNING: Peer does not respond to virsh (${p})" && HA_WARNINGS=$(( ${HA_WARNINGS}+1 )) && continue
  n=$(( ${n}+1 ))
done
[ ${n} -eq 0 ] && echo "ERROR: No peers are reachable" && HA_ERRORS=$(( ${HA_ERRORS}+1 ))

# Check stonith devices
if [ "${HA_HOST:0:5}" == 'janus' ]; then
  STONITH_DEVS='mfsys:10.17.20.0'
  STONITH_OPTS='server1_hostname=test'
else
  STONITH_DEVS='ibmbc:10.17.21.251 ibmbc:10.17.22.251 ibmfx:10.17.31.253 ibmfx:10.17.32.253'
  STONITH_OPTS=''
fi
for d in ${STONITH_DEVS}; do
  echo "INFO: Testing stonith device '${d}'"
  t="${d%:*}"; ip="${d#*:}"
  eval "stonith -s -S -t external/${t} mgmt_address=${ip} snmp_conf_path=/havc/config/stonith/${t} ${STONITH_OPTS}" >/dev/null
  [ $? -ne 0 ] && echo "ERROR: Stonith device does not respond to status query (${d})" && HA_ERRORS=$(( ${HA_ERRORS}+1 ))
done

# Done
if [ ${HA_WARNINGS} -eq 0 -a ${HA_ERRORS} -eq 0 ]; then
  echo 'SUCCESS! You can safely enable HA.'
elif [ ${HA_ERRORS} -eq 0 ]; then
  echo 'WARNING! Not all tests pass; you can enable HA, but are you sure?'
else
  echo 'ERROR! Critical tests failed; do NOT enable HA!'
fi
```

/havc/scripts/havc-shell

```
#!/bin/bash
# Cedric Dufour <cedric.dufour@idiap.ch>


## Usage
[ $# -lt 1 -o "${1##*-}" == 'help' ] && cat << EOF && exit 1
USAGE: ${0##*/} <command> [<...>]

SYNOPSIS:
  Wrapper for (supported) Pacemaker/Libvirt commands:
   ha-..., havc-... (HAVC)
   locate, crm (crm)
   domstate, start, shutdown, destroy, console, vncdisplay (virsh)
   top (virt-top)
EOF

# Arguments
MY_COMMAND="${1}"; shift


## Functions

# Locate the given Libvirt domain (aka. Pacemaker resource)
function __crm_locate {
  MY_DOMAIN="${1}"
  [ -z "${MY_DOMAIN}" ] && echo "ERROR: Missing domain argument" >&2 && return 1
  MY_NODE="$(ssh admin.havc "crm_resource --locate --quiet --resource ${MY_DOMAIN}")"
  [ -z "${MY_NODE}" ] && echo "ERROR: Unable to locate resource/domain" >&2 && return 1
  echo "${MY_NODE}"
}


## Main
case "${MY_COMMAND}" in

  'locate')
    MY_DOMAIN="${1}"; shift
    MY_NODE="$(__crm_locate "${MY_DOMAIN}")"
    e=$?; [ ${e} -ne 0 ] && exit ${e}
    echo "INFO: '${MY_DOMAIN}' is running on HAVC node '${MY_NODE}.havc'"
    exit 0
    ;;

  'domstate'|'start'|'shutdown'|'destroy'|'vncdisplay')
    MY_DOMAIN="${1}"; shift
    MY_NODE="$(__crm_locate "${MY_DOMAIN}")"
```

```
    e=$?; [ ${e} -ne 0 ] && exit ${e}
    MY_SSH_OPTIONS=''
    echo "INFO: virsh --connect qemu://${MY_NODE}/system ${MY_COMMAND} ${MY_DOMAIN}" >&2
    o="$(eval ssh ${MY_SSH_OPTIONS} ${MY_NODE}.havc "virsh ${MY_COMMAND} ${MY_DOMAIN}")"
    e=$?
    e=$?; [ ${e} -ne 0 ] && exit ${e}
    [ "${MY_COMMAND}" == 'vncdisplay' ] && o="${MY_NODE}.havc${o}"
    echo "${o}"
    exit 0
    ;;

  'console')
    MY_DOMAIN="${1}"; shift
    MY_NODE="$(__crm_locate "${MY_DOMAIN}")"
    e=$?; [ ${e} -ne 0 ] && exit ${e}
    MY_SSH_OPTIONS=''
    echo "INFO: virsh --connect qemu://${MY_NODE}/system ${MY_COMMAND} ${MY_DOMAIN}" >&2
    eval ssh -t ${MY_NODE}.havc "virsh ${MY_COMMAND} ${MY_DOMAIN}"
    exit $?
    ;;

  'qemu-monitor-command')
    MY_DOMAIN="${1}"; shift
    MY_NODE="$(__crm_locate "${MY_DOMAIN}")"
    e=$?; [ ${e} -ne 0 ] && exit ${e}
    MY_SSH_OPTIONS=''
    echo "INFO: virsh --connect qemu://${MY_NODE}/system ${MY_COMMAND} ${MY_DOMAIN} --hmp $@" >&2
    eval ssh -t ${MY_NODE}.havc "virsh ${MY_COMMAND} ${MY_DOMAIN} --hmp $@"
    exit $?
    ;;

  'crm')
    ssh -t admin.havc 'crm'
    exit $?
    ;;

  'top')
    MY_DOMAIN="${1}"; shift
    MY_NODE="$(__crm_locate "${MY_DOMAIN}")"
    e=$?; [ ${e} -ne 0 ] && exit ${e}
    ssh -t ${MY_NODE}.havc 'virt-top'
    exit 0
    ;;

  'havc-enable')
    echo "ERROR: Shamelessly refusing to wrap this command (${MY_COMMAND})" >&2
    echo "ERROR: Please invoke it directly (not via ${0##*/})" >&2
    exit 1
```

```
    ;;

  'ha-'*|'havc-'*)
    ssh -t admin.havc "${MY_COMMAND} $@"
    exit $?
    ;;

  *)
    echo "ERROR: Unsupported command (${MY_COMMAND})" >&2
    exit 1
    ;;

esac
```