AlphaVM for Unix
User Manual

Date: 28-Feb-2020

Author: Artem Alimarin

Version: 1.5.67

TABLE OF CONTENTS

1 Scope

This is a user manual for the AlphaVM line of products currently represented by AlphaVM-Pro and AlphaVM-DC running on variants of Unix.

AlphaVM-Pro, the professional Alpha system emulator. It is meant to replace Alpha servers working in data centers or industrial settings. It has high performance and reliability characteristics. It's performance is on the level of the real Alpha systems. AlphaVM-Pro is capable of replacing machines of DS10, DS20, ES40, DS25, ES45 class of Alpha systems.

AlphaVM-DC is a variant of AlphaVM-Pro for data centers. The main difference is in the licensing. AlphaVM-DC license binds to the hypervisor appliance virtual hardware rather than to a USB dongle.

AlphaVM-Basic is a basic emulator that supports 1 basic CPU and 1GB RAM.

2 Installation

2.1 Host platform requirements

2.1.1 Requirements for AlphaVM-Pro

The general requirements are as follows:

- AlphaVM is currently supported following host systems:
    - Linux RHEL CentOS 6 and 7.
    - Linux Debian 8, 9, 10.
    - FreeBSD x64 9,10, 12.
- The host system must have a CPU, which supports CMPXCHG16B instruction, SSE3 and SSSE3 instruction set. Almost all 64-bit CPUs support the instructions except for an older AMD Opteron. For instance, Intel Core2 supports them.
- The host CPU performance has a direct influence on the emulated system performance. The actual host CPU type depends on your performance requirements. We recommend fast new Intel CPUs (at least 3GHz). Here are some of them
    - E3-12xx v5 (4 cores) with 3.5GHz or faster,
    - E5-26xx v4 (6 cores), 3.5GHz or faster,
    - E5-26xx v4 (4 cores), 3.5GHz or faster, the host system can have 1 or 2 CPUs.

  HP ProLiant servers can be equipped with these CPUs.
- The product requires a reserved host CPU core for each emulated CPU. This means that a single CPU emulator requires at least a dual-core host system. A dual CPU emulator requires a 3 core system. We advise a double number of cores with respect to the emulated CPUs. Lack of CPU resources can cause not only performance degradation, but also CPU sanity checks in OpenVMS, which are caused by emulated CPU unavailability.

- The host memory requirement depends on the emulated Alpha memory size and other emulator settings. The general requirement is as follows: if the emulated system has N GB memory, than the host system must have at least N + 2GB memory. Besides the emulated memory size, the following factors may influence the required amount: number of CPUs, number of disks, caching settings for disks. We advise N + 4GB.
- Emulated Ethernet controllers are mapped to host Ethernet adapters. Note that mapping to Wireless Ethernet controllers does not always work. It is advisable to have a separate host NIC for each emulated NIC.
- JIT CPU versions require more memory and CPU resources than the basic CPU.
- The product is supported on virtual platforms: Hyper-V, VMware and Proxmox VE. The product may also run on other platforms like Virtual Box. However, running on a hyper-visor means extra level of virtualization, which can cause performance degradation. This document does not cover in detail the configuration specifics for these platforms. The details can be found on our website.
- Emulated Ethernet controllers are mapped to host Ethernet adapters. Note that mapping to Wireless Ethernet controllers does not always work. It is advisable to have a separate host NIC for each emulated NIC.

The actual requirements are very specific for a given configuration and workload. Contact our support to get an advice about hardware required for your configuration.

2.1.2 Requirements for multiple instances

It is possible to run multiple instances of the emulator on the same machine. The requirements in this case just add up.

For instance, if you run two similar virtual machines on the same host, the requirements are double of the single instance requirements.

2.1.3 Requirements for running on a hyper-visor.

AlphaVM is an application that runs on the host OS. The host OS in turn can run on a physical hardware or on virtual hardware provided by VMware, Hyper-V, Proxmx VE, etc.

The requirements for the hosting VM are the same as for a hardware host requirements.

It is recommended to use a dedicated host VM NIC for each AlphVM NIC.

When running on a hypervisor, the hypervisor controls how the host hardware system resources are assigned to the VM that host AlphaVM. It is important that AlphaVM gets enough resources to show adequate performance. AlphaVM performance will suffer on an overcommitted hypervisor.

The AlphaVM CPU emulator for each Alpha CPU has a single main thread that implements the main CPU workflow. This thread either interprets Alpha instructions read from memory or executes native code created by the just-in-time compiler for the Alpha instruction stream. Alpha CPU has more auxiliary threads that, for instance, run the just-in-time compiler. However, the main CPU workload is inherently

single threaded, because the CPU actions are essentially sequential, although in real CPU some fine grained parallelism could occur due to pipelining. In AlphaVM the instructions are executed sequentially.

The single-threaded nature of Alpha CPU means that a single Alpha CPU is not scalable and it needs 100% of the host CPU to execute efficiently. If Alpha CPU gets, for instance, only 50% of the host CPU power, Alpha CPU performance will be about 50% of what it could be.

Alpha CPU constantly keeps executing the Alpha instruction stream, which explains 100% host CPU resource usage. The only reason why Alpha CPU can stop using 100% of the host CPU resources is the idle release – a feature that allows to release the host CPU when the guest OS (OpenVMS or Tru64) is idle. This is achieved by the guest OS idle loop detection feature.

Naturally, different Alpha CPUs can run in parallel on different host cores.

It is recommended for the VM hosting AlphaVM to be given enough resources to ensure that AlphaVM always gets 100% of the host CPU and enough memory backup.

If AlphaVM performance is important, we recommend running on a physical server rather than on a hypervisor.

2.2 Obtaining the software

Please contact us per e-mail

mailto://sales@emuvm.nl

AlphaVM-Pro requires a USB license key to run. The key and the software will be sent to you when you purchase the software.

AlphaVM-DC requires a software key to run. We generate the license key based on the information about hypervisor appliance virtual hardware provided by our tool.

The software license price depends on the virtual Alpha system configuration. You can request a quote on the page http://emuvm.com/alphavm_pro_quote.php.

2.3 Installation procedure

Currently the product is delivered as

- *.tgz file for Linux and FreeBSD
- *.rpm file for RedHat, CentOS and other Linux flavors that have rpm packaging system.
- *.deb for Debian and other Linux distributions that have Debian packaging system.

The software is linked against the system libraries dynamically. It imposes some dependencies on the hosting OS environment. The RPM and DEB packages check the requirements.

2.3.1 Installing from TGZ

Installation example:

```
root@debdell:/emuvm# tar xvzf alphavm-pro-1.5.41.Debian7.x86_64.tgz

root@debdell:/emuvm# cd alphavm-pro-1.5.41/

root@debdell:/emuvm/alphavm-pro-1.5.41# ./install.sh
```

Example of uninstallation:

```
root@debdell:/emuvm/alphavm-pro-1.5.41# ./uninstall.sh

cd ..

root@debdell:/emuvm# rm -fr alphavm-pro-1.5.41
```

2.3.2 Installing from DEB package.

Use the usual dpkg tool to install and uninstall the package

```
dpkg -i alphavm-pro-1.5.41.Debian7.x86_64.deb
```

Uninstall like this:

```
dpkg -r alphavm-pro
```

Some files are installed to /usr/share/alphavm-pro.

2.3.3 Installing from RPM package.

Use the usual rpm tool to install and uninstall the package

```
rpm -i alphavm-pro-1.5.41.CentOS7.x86_64.rpm
```

Uninstall like this:

```
rpm -e alphavm-pro
```

Some files are installed to /usr/share/alphavm-pro.

2.4 Installed files

Here is the description of the installed files:

- The professional virtual machine **alphavm_pro**. Installed to /usr/bin/alphavm_pro
- The example configuration file **example.emu**. This file can be copied and modified according to your needs.
- The **README** file contain the main product information
- The product release notes **relnotes.html**.
- The product user manual **usernaual.pdf**. It is this user manual.
- The empty disk image creator **emuvm-mkdisk**. Use this utility to create empty disks images to be used with the emulator.

- The license service **emuvm-keylok**. This executable is responsible for checking the license dongle (not on FreeBSD).
- The VM starter service **emuvm-server**. The executable is responsible for automatic VM startup and restart.

The RPM and DEB installers install some files to /usr/share/alphavm-pro. The install.sh procedure from TGZ does not install to /usr/share. The files can be found in the directory extracted from TGZ.

2.5 Permissions to access Ethernet

In order to use network the emulator has to run either as **root**.

On Linux it is possible to use capabilities to give a non-root user access to the raw Ethernet device. The Linux capabilities are CAP_NET_ADMIN and CAP_NET_RAW. If the emulator accesses the network without permissions, it may crash with SIGSEGV.

2.5.1 Running as root

Running as root can be achieved by running with **sudo** or by using **setuid**.

2.5.2 Setting capabilities on Linux

To set the capabilities please execute the following or similar commands as root:

```
apt-get install libcap2-bin

setcap cap_net_raw,cap_net_admin=eip alphavm_pro
```

Check that the capabilities are set:

```
getcap alphavm_pro
```

It prints:

```
alphavm_pro = cap_net_admin,cap_net_raw+eip
```

2.6 Checking the status of a KEYLOK dongle on Linux

The professional version of the emulator uses a USB dongle for its protection. The software does not run without the dongle. The dongle vendor is KEYLOK.

You can use **lsusb** command to check that the dongle is present. The dongle is reported as follows:

```
Bus 005 Device 002: ID 0471:485e Philips (or NXP)
```

The **lsusb –v** command prints detailed information.

If you do not have the **lsusb** command, install it as follows:

```
apt-get install usbutils
```

2.7 Running the KEYLOK dongle service on Linux

The evaluation version uses remote EmuVM server for licensing. In this case the dongle is not used and license service does not have to be started.

If you have the dongle, plug it to the system where you want to have the dongle and start keylok_service on that machine. Later configure your emulator to use that machine's IP address for licensing.

Usually the dongle is connected to the same machine that host AlphaVM. The service also runs on this machine. Localhost or 127.0.0.1 can be used in the AlphaVM licensing configuration.

2.8 Install a terminal emulator

You need a terminal emulator to be able to communicate to the emulator Alpha console. We recommend PuTTY. Alternatively, you can connect any other terminal emulator.

2.8.1 Install PuTTY on Debian

To install PuTTY do the following.

```
apt-get install putty
```

See section about serial line configuration for information about configuring PuTTY.

3 Configuration

The emulator must be configured before it can be used. The configuration specifies properties of the emulated system.

The emulator configuration is defined by means of a configuration file. See the configuration file example.emu included in the distribution. We recommend to copy and edit this configuration file according to your needs.

The configuration file defines the emulated system configuration and how the emulated devices are mapped to entities of the host machine.

3.1 System configuration

System configuration screen enables configuration of the emulated Alpha system.

Configuration properties:

- The option **type** specifies the type of the emulated system. Currently the following systems are supported:
  - ds10_466 – AlphaServer DS10 466MHz, model 1839
  - ds10_616 – AlphaServer DS10 616MHz, model 1970
  - ds10l_466 – AlphaServer DS10L 466MHz, model 1961
  - ds10l_466 – AlphaServer DS10L 616MHz, model 1962

- ds20_500 – AlphaServer DS20 500MHz, models 1839, 1920
- ds20e_500 – AlphaServer DS20E 500MHz, models 1840, 1921
- ds20e_667 – AlphaServer DS20E 667MHz, models 1939, 1940
- ds20e_833 – AlphaServer DS20E 833MHz, models 1982, 1983
- ds20l_833 – AlphaServer DS20L 833MHz, model 2006
- es40_500 – AlphaServer ES40 500MHz, models 1813 -1816
- es40_667 – AlphaServer ES40 500MHz, models 1817 -1820
- es40_833 – AlphaServer ES40 833MHz, models 1984 - 1987
- xp900_466 – AlphaStation XP900 466MHz, model 1879
- xp1000_500 – AlphaStation XP900 500MHz, model 1821
- xp1000_667 – AlphaStation XP900 667MHz, model 1822
- xp1000_750 – AlphaStation XP900 750MHz, model 1922

- The options reported_type specifies the type of the system reported to the OS informational routines (like licensing). This option allows specifying systems that are not actually implemented. By default the same system information is returned as specified by **type**. The value **default** instruct the system to use the same type as the specified by **type**. Other values are:
  - as4000_667,
  - as4100_667,
  - ds10_466,
  - ds10_616,
  - ds10l_466,
  - ds10l_616,
  - xp900_466,
  - xp1000_500,
  - xp1000_667,
  - xp1000_750,
  - ds20_500,
  - ds20e_500,
  - ds20e_667,
  - ds20e_833,
  - ds20l_833,
  - es40_500,
  - es40_667,
  - es40_833,
  - ds25_933,
  - es45_1000.
  - Other systems will be added on request.
- The option **num_cpus** specifies the number of CPUs in the emulated system. The emulator reserves one core of the host system for each emulated CPU. The emulator needs at least one core for some bookkeeping and IO processes. Thus, you need a dual core system to run with one emulated CPU, and at least 3-cores to run a dual CPU configuration. The maximal number of

CPUs depends on the emulated system and on the product license. The default number of CPUs is 1.

- The option **ssn** specifies the system serial number of the emulated system. SSN is often used by third party software to identify the hardware for licensing purposes. The default value is empty string. This value is a string of max 16 characters long.
- The option **interval_clock_freq** specifies the interrupt clock frequency in Hz. Interrupt clock frequency specifies the number of timer interrupts per second. Please do not change this value unless you know what you are doing. This value can affect stability of the system. The standard Alpha frequency is 1000 interrupts/second. However, this frequency can be changed for performance tuning reasons. It is communicated to the operating system via HWRPB. OpenVMS and Tru64 adjust to this value. For performance reasons, it could be better to set this value to 100. Currently Linux does not seem to work correctly with non-standard values.
- The option **clock_busy_wait** specifies whether to use a busy loop to make the clock intervals precise. The default value is True, which corresponds to the behavior of the product prior to introduction of this option. This option is introduced to prevent excessive CPU usage by the timer when the host is not capable of providing reliable intervals. Disabling busy loops can be useful when running on a host with scarce CPU resources or on a virtual host that is poorly scheduled.
- The option **adjust_clock_resolution** specifies whether AlphaVM tries to change the hosting OS clock resolution to have a better response time on Windows level. By default, it is true.
- The option **cycle_counter_freq** can be used to override the default cycle counter frequency defined by the emulated system. The default value for this option is zero, which means that the default clock of the emulated system is used. This option does not influence the real performance of the VM. However, this option can be useful when an application in the guest system uses a timing calibration algorithm based on the cycle counter frequency.

3.2 CPU Configuration

CPU configuration node is used to configure all CPUs in the system, which have similar properties.

Configuration properties:

- The option **server** specifies a CPU server to be used. Currently there are there servers available:
  - The **basic** server is a server with a basic performance.
  - The **jit1** server is a server with the performance on the level of fast EV4. It is based on JIT technology.
  - The **jit2** server is a server with the performance on the level of high-end EV5 – low end EV6 Alpha CPUs.
  - The **jit3** server is the fastest server with the performance of high-end EV6 – EV7 Alpha CPUs. Its performance is approximately double of **jit2**.
- The option **stat_period** specifies the period between the CPU statistics dumps. For the workload profiling purposes the CPU can dump statistics over a time period. The default value is zero, which means that the dumps are disabled. Do not turn this option on for a production system.

- The option **tbchk** specifies whether PAL TBCHK instruction is implemented. By default it is implemented.
- The option **suppress_unaligned** specifies whether unaligned trap is to be generated when applicable; true by default. For some workloads that generate a lot of unaligned accesses it could be desirable to disable the unaligned access traps to increase performance. In that case AlphaVM performs unaligned access fixup in the similar way to the unaligned trap handler of OpenVMS or Tru64.
- The option **pedantic_ieee_fp_traps** specifies whether IEEE FP trapping is implemented exactly according to the Alpha architecture specification. It is by default true. When false, some more performance optimizations are possible. When disabled, it the inconsistency with Alpha architecture affects only trapping cases.
- The option **pedantic_sfloat_rnd** specifies whether rounding in IEEE single-precision floating point computations is implemented exactly according to the Alpha architecture specification. It is true by default. When disabled, some performance optimizations are enabled.
- The option **queue_lock_retries** specifies the number of retries when trying to acquire the queue lock in the OpenVMS interlocked queue PALcode instructions. The default value is zero, which means that the system chooses the value.
- The options **queue_lock_spins** specifies the number of spins when trying to lock a queue in the OpenVMS interlocked queue PALcode instructions. The default value is zero, which means that the system chooses the value.

JIT configuration:

- The option **async** (the values are **yes** or **no**) specifies whether JIT compilation process is synchronous with respect to the CPU or asynchronous. By default it is asynchronous (**yes**). The synchronous mode is needed mostly for debugging.
- The option **idle** enables the emulator to release the host CPU when the guest OS is idle. This feature can cause performance degradation of some IO-loads. This feature is currently experimental and is available for field testing only.
- The option **experimental** enables some experimental optimization features that have field test status.
- The option **max_pages** specifies the maximal number of JIT pages that can be simultaneously active in the system. Each JIT page corresponds to a single page of Alpha code. Please do not change unless you know what you are doing.
- The option **code_size** specifies the default size in KB for memory allocation of code chunks. The default is 256k. Too big chunks can cause excessive memory consumption. Too small chunks can degrade the system performance due to frequent allocation. The advised values are in range 128 - 1024. The value is rounded up to 64k.
- The option **imb_mode** - this is an advanced feature for performance tuning. Do not change it unless advised to do so by EmuVM.
- The option **imb_on_rei** – this is an obsolete feature that enables automatic instruction memory barrier on PAL REI. Do not enable it.

- The option **host_fp_traps** specified whether the FP traps are implemented using the host FP traps or by checking conditions. It is false by default, which corresponds to the old behavior. Setting it to true can yield a drastic performance improvement for some FP intensive workloads.
- The option **num_threads** specified the number of just-in-time compiler threads used to compile the Alpha instruction stream into native code. This number specifies the number of threads per AlphaVM CPU, rather than in total. The default is zero, which means that AlphaVM selects the appropriate value automatically. Currently AlphaVM sets it to one. The value currently could be 0, 1, 2 or 4. The other values are converted to one of these values. Setting more than one thread can improve some workloads that cause a lot of JIT compilation. The host machine must have enough CPU power to benefit from additional threads.
- The option **queue_lock_spin** is for performance tuning by EmuVM engineers. It specified the number of attempts the CPU tries to acquire the JIT queue lock. Actually, **queue_lock_spin** + 1 attempts are performed. The default value is zero, which means that only a single attempt is made. When all attempts are exhausted, the action depends on **queue_lock_block**. The CPU is either blocked until it can acquire the lock or the request is dropped.
- The option **queue_lock_block** is for performance tuning by EmuVM engineers. Specifies whether the CPU is blocked or the request is dropped when **queue_lock_spin** attempts to lock the JIT queue are exhausted. The default is not to block. Blocking the CPU can lead to performance problems.
- The option **queue_full_spin** is for performance tuning by EmuVM engineers. It specified the number of attempts the CPU tries to push a JIT request onto the JIT queue. Actually, **queue_full_ spin** + 1 attempts are performed. The default value is zero, which means that only a single attempt is made. When all attempts are exhausted, the action depends on **queue_full_block**. The CPU is either blocked until the request can be pushed or the request is dropped.
- The option **queue_full_block** is for performance tuning by EmuVM engineers. Specifies whether the CPU is blocked or the request is dropped when **queue_full_spin** attempts to lock the JIT queue are exhausted. The default is not to block. Blocking the CPU can lead to performance problems.

3.3 Memory configuration

Configuration properties:

- The option **size** specifies the RAM size in megabytes of the emulated system. The amount of memory you can use here depends on the amount of memory on your host computer. It is recommended to have at least 2GB of host memory. Maximal memory size depends on the emulated system and on the product licensing. The default size is 128M.
- The option **lock** specifies whether to lock the guest memory in the host memory. Locking means that the pages are not unloaded from the memory by swapping. Locking can degrade performance, because other pages will be offloaded. This is an advanced option added for experimentation purposes. Locking is by default off. Note that you can only lock pages when the

working set is large enough. Additionally you may need to set special capabilities to allow locking. If the VM fails to lock pages and logs an error and continues without locking.

3.4 SCSI Controller configuration

A SCSI controller can be loaded by sections like

```
scsi_controller qla0 {
        scsi_id = 7;
}
```

The number of loaded SCSI controllers determines the number of available SCSI buses. Currently we emulate only QLOGIC ISP 1020 SCSI Controller.

Some Alpha systems have one or two built-in SCSI controllers. These controllers are loaded automatically. In this case, the configurations qla0 and qla1 specify configuration of those pre-loaded controllers, instead of loading a new controller. It is recommended to have explicit configuration for such built-in controllers.

Configuration properties:

- The option **scsi_id** specifies the SCSI ID of the controller.
- The option **slot** specifies the PCI slot to which the adapter is plugged. The default behavior is automatic; thus, you do not have to specify this option or know about slots. The automatic behavior covers most cases with OpenVMS. Tru64, however, is very sensitive to changes in the hardware configuration. When you copy your disk images from the real system, it can be required to specify the slots in such a way to reflect the configuration of the real. The number of slots depends on the actual emulated system. The mapping of the slots to PCI hoses and IDSELs also depends on the system.

3.5 Disk configuration

New disks can be added in the Configuration menu. It can be removed or renamed by means of a context sensitive menu available on right-click on the device in the configuration tree. Note that the device name has no meaning for the system, it is only used for convenience. For instance, you can choose it to be the same as your disk label or with the disk name in the SRM or in your guest OS.

The disk image must exist before you can attach it to the emulator. A fresh disk image can be created by means of the Make Disk tool available in the Tools menu. The Make Disk tool just creates an empty disk image file. It does not attach it to the emulator. Therefore, you have to attach it yourself after creation.

Disk configuration table enables configuration of the emulated disk properties.

Configuration properties:

- The options server specifies how the disk is server. By default AlphaVM chooses it automatically. The supported values are the following:
    - **default** – the system chooses a server. It is **aio** except for CDROMs, where it is **basic**.
    - **basic** – server based on the standard unix IO.
    - **aio** – server based on asynchronous IO.
    - **mapped** – server based on memory-mapped files. It is only supported when the storage is a regular file.
- The option **file** specifies a file name of the disk image file used to store disk data. An empty disk image can be created using the **mkdisk** utility provided in the package. Please note that creation of a disk image does not connect it to the system. After creation, you still need to specify it in the **file** property of one of the disks. There is no default value, the value must be provided. The value can also be a disk block device.
- The option **scsi_bus** specifies a SCSI bus to which the device is connected. The buses are numbered from zero. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. The default value is zero.
- The option **scsi_id** specifies the SCSI target ID of the disk. The SCSI ID can have values 0 .. 6, 8 - 15. SCSI ID 7 is reserved for the SCSI controller. All SCSI devices on each bus must have unique SCSI IDs and LUNs. The default value is 0.
- The option **scsi_lun** specifies SCSI logical unit of the disk device. The value can be 0..7. SCSI. This option allows several logical devices to be associated with a single bus device. The default is 0.
- The option **async** enables asynchronous operation of the disk with respect to the SCSI controller. It is by default **yes**. It can make sense to turn it off for very fast memory-mapped IO, when extra context switches may cause extra overhead.
- The option **removable** has effect for disk images based on regular files. It is **no** by default. If it is **yes**, the emulator returns the status “no medium” instead of “offline”, when the file is not present and cannot be opened. The emulator also handles load/unload medium commands in such a way that the image is opened/closed. The user can replace the image while it is closed. Unfortunately OpenVMS does not send the unload command when the disk is dismounted with even with the /UNLOAD qualifier. Use the following commands to unload the removable medium:

    ```
    $ rzt:==$sys$etc:rztools_alpha
    $ rzt dka100: /stop
    ```

    On Tru64 you can use the following command to eject the removable medium:

    ```
    scu –f /dev/rdisk/cdrom0c eject
    ```
- The option **caching** specifies whether caching of the disk image file is enabled on the host operating system level. The default value is **no**. This option corresponds to *O_DIRECT* flag of the *open()* system call.
- The option **write_through** specifies whether write-through mode is enabled on the host operating system level. The default value is **no**. This option corresponds to *O_SYNC* flag of the *open()* system call. Write through mode can result in a significant performance degradation.

- The option **shared** specifies whether the VM opens the disk image in shared mode. Normally it should be opened in exclusive mode to prevent multiple usage of the same file. The default value is **no** (exclusive mode), which guarantees that the disk can be modified only by the emulator.
- The option **read_only** specifies whether the emulated disk is read-only. In this case the VM opens the image in read-only mode. The default value is **no** (writable)
- The options **trace_sense** enables logging of commands with sense data. Normally sense data is associated with errors or non-standard situations, so you may wish to enable it to see if something is going wrong. The default value is **no**.
- The options **vendor, product,** and **revision** specify the emulated disk attributes. When these attributes are unset, the AlphaVM provides some default attributes.
- The option **vendor_specific** specifies the vendor specific field in the VPD field.
- The option **pr_mode** specifies how the SCSI persistent reservations are implemented.
    - **none** – means that persistent reservations are not implemented and the disk returns the status "invalid command" for these SCSI commands.
    - **dummy** – means that the device implements the commands, but no actual protection of reservations is done. The implementation is dummy. This option can be used when there is just one node working with the disk. It is useful for a single node Tru64 cluster.
    - **real** – the system implements persistent reservations.

  *Currently only **none** is supported.*
- The option **vpd** enables SCSI vital product data reporting. The default is currently false.
- The option **device_eui64** specifies the device indetifier in the EUI-64 form (8,12 or 16 bytes). This form is provided by some SCSI or FibreChannel disk arrays. Example: *0000-0E11-0012-5205*. It corresponds to Tru64 *SCSI-WWID:0c000008:0000-0e11-0012-5205*.
- The option **device_scsi_name** specifies the device identifier in the SCSI name form.
- The option **device_vid** specifies the device identifier (VPD page 83). Example: *DEVICE-VID-EMUVM-0001.* It corresponds in Tru64 to something like *SCSI-WWID:03100025:"RZ26L DEVICE-VID-EMUVM-0001".*
- The option **device_naa** specifies the device identifier in the NAA form (8 or 6 bytes). Example: *6000-1fe1-0010-8d40-0001-0460-7270-00ca*. This corresponds to Tru64: *SCSI-WWID:01000010:6000-1fe1-0010-8d40-0001-0460-7270-00ca*.
- The option **device_sn** specifies the device serial number (VPD page 80): Example: DEVICE-SN-EMUVM-0001. In Tru64 it corresponds to something like *SCSI-WWID:0410002c:"DEC RZ26L DEVICE-SN-EMUVM-0001"*. The device serial number can also be used to emulate a HSZ unit. To emulate HSZ the vendor must be *DEC* and the product must start with *HSZ*. In this case the device SN is a concatenated SNs of this HSZ and the other HSZ in the dual set (20 characters together, 10 for each SN). Thus, *ZG41000118ZG41000119* corresponds to Tru64 *SCSI-WWID:0910003c:"DEC HSZ70 ZG41000118ZG41000119:d00t00003l00000"*
- The option **port_eui64** specifies the device port EUI-64 identifier.
- The option **port_naa** specifies the port NAA identifier.
- The option **port_scsi_name** specifies the port name in the SCSI form.

- The option **trace_cmd** – enables tracing of SCSI commands
- The option **trace_sense** – enables tracing of SCSI sense information. SCSI sense information is normally send when an error or a non-standard situation occurs.
- The option **trace_io** – enabled tracing on the level of disk server.

3.5.1 Disk names in AlphaVM SRM console

The SRM or VMS disk device name, e.g. dkb1201, is formed as follows:

- The First two letters **dk** designate SCSI disk
- The third letter designate the SCSI controller number **a**=0, **b** =1, …
- The number **n** defines SCSI id and logical unit: id=**n**/100, lun = **n** % 100

Thus dkb101 means that the disk is connected to the bus of the second SCSI controller (bus=1), SCSI ID is 12, SCSI LUN is 1.

3.5.2 Performance considerations

For most workloads it is recommended to use asynchronous IO with caching off and write through off.

Caching causes performance degradation for large disks and some operations like large file copies. In this case it causes excessive swapping the host OS level.

Caching can be beneficial when the disk is relatively small comparing to the host RAM.

Disk IO performance depends on multiple factors. The following is to consider when tuning IO performance.

- Adjust AlphaVM process working set size. The working set limits can be set in the Launch configuration section. The maximal working set specifies *rlimit* for *RSS*.
- Adjust Linux *swappiness*. This setting can be reduced to 0 or 10.
- Disable swapping. In a tuned dedicated AlphaVM host swapping should not be needed. If it is really needed, there probably is not enough host resources.

3.6 CDROM configuration

CDROM configuration is similar to disk configuration. The default server for an ISO image is **aio**. The default server for a CDROM device is **basic**.

CDROM does not have write-related properties. ISO images are always opened in read-only mode.

If you wish to access the physical CDROM in the host system, specify the block device name like in the **file** option.

On Linux CDROM is a block device called something like /dev/cdrom.

On FreeBSD CDROM is a character devices called like /dev/cd0 or /dev/acd0.

3.7 SCSI Tape configuration

AlphaVM supports virtual (logical) SCSI tapes. The tape is emulated using a tape image file. Note that physical tapes are supported only on AlphaVM-Pro by means of SCSI Pass through, described in the next section.

As with other SCSI devices, the SCSI path should be unique.

Currently the virtual tape drive has no button or emulator command to load/unload the medium while the emulator is running. On OpenVMS please use rztools:

```
$ rzt:==$sys$etc:rztools_alpha
```

Send load command to the tape:

```
$ rzt mka600: /start
```

Send unload command to the tape:

```
$ rzt mka600: /stop
```

On Tru64 the tape can be operated with:

```
# scu -f /dev/rmt0h
```

Configuration properties:

- The option **file** specifies a file name of the tape image file used to store data. An empty tape image can be created by creating an empty file. The default value is empty, which means that there is no medium in the tape drive.
- The options **scsi_bus, scsi_id, scsi_lun** are similar to the same options for SCSI disks.
- The option **async** enables asynchronous operation of the tape with respect to the SCSI controller. It is by default **yes**. Tape it is a very slow device, which can block IO when in synchronous mode. This option is provided for debugging.
- The option **initial_load** specifies whether the tape medium is loaded in the drive when the emulator starts. This is applicable only if the tape image file exists. If the image does not exist, it is considered that there is no medium in the drive.
- The option **auto_load** specified with the tape is automatically loaded on access. This means that the tape file is opened on access. When this option is off, a special load command must be issued to load the tape (see *rztools* commands earlier in this section). When auto-load is on, you do not need those commands. Note that multi-volume backups do not work with auto-load, because you do not have a chance to swap the media: the tape will automatically reopen the same file when it is done with the first volume.
- The option **auto_create** specifies whether an empty tape file created if it does not exist. It is convenient; because you do not have to create empty tape files yourself.

- The option **max_size** specifies the maximal size of the tape image file. This parameter can be used to create a multi-volume tape backup. The default value is zero, which means no limit.
- The option **read_only** can be used to protect the tape from writing.
- The option **shared** indicates the shared open mode of the tape drive.

3.8 SCSI Pass-Through (aka SCSI direct, aka Generic SCSI) configuration

AlphaVM-Pro is capable of accessing the host system SCSI devices by means of so called SCSI Pass Through mechanism. SCSI commands and data are passed between the guest and the hosts systems "as is". This feature allows to access devices that are not available via the emulation layer. Examples of where the SCSI Pass Through is useful include access to the following devices:

- SCSI tape, which is not available otherwise
- SCSI disk, which can be taken from the real system and attached to the emulator to simplify migration
- ATAPI CDROM, which works as expected
- iSCSI disks
- SCSI devices of other types

Although the intentions of SCSI Pass Through mechanism is to pass commands and data "as is", some options are available to adjust commands and data in such a way that some useful devices are not rejected by OpenVMS or Tru64.

Configuration properties:

- The option **scsi_bus** specifies a emulated SCSI bus to which the device is connected. The buses are numbered from zero. The number of SCSI buses depends on the amount and type of the SCSI controllers in the system. The default value is zero.
- The option **scsi_id** specifies the emulated SCSI target ID of the disk. The SCSI ID can have values 0 .. 6, 8 - 15. SCSI ID 7 is reserved for the SCSI controller. All SCSI devices on each bus must have unique SCSI IDs and LUNs. The default value is 0.
- The option **scsi_lun** specifies the emulated SCSI logical unit of the disk device. The value can be 0..7. SCSI. This option allows several logical devices to be associated with a single bus device. The default is 0.
- The option **async** enables asynchronous operation of the unit with respect to the SCSI controller. It is by default **yes**. A slow device in a synchronous mode can block IO. This option is provided for debugging.
- The option **device** specifies the Linux pass through device used as the backend. For example, it can be /dev/sg3 (Linux) or /dev/pass3 (FreeBSD). For enabling and naming of Linux and FreeBSD pass devices see the following subsections.
- The option **suppress_vpd** is used to suppress the Vital Product Data returned by the SCSI Inquiry command. Such data is often is incompatible with Tru64, which causes the device to be rejected. When the data is suppressed, as if VPD is not supported by the device, Tru64 is happy to use the device. The default value is **no**.

- The option **emulate_mp1** enables emulation of the SCSI mode page 1 (read-write error recovery page) to be emulated. This option is needed because OpenVMS rejects disks that do not implement this mode page. The default value is **no**.
- The option **convert_cdb** enables conversion of 6-byte SCSI commands to 10-byte SCSI commands. This mode can be used to access ATAPI devices, which understand 10-byte SCSI commands. The default value is **no**.
- The option **check_sn** specifies a SCSI device serial number. When AlphaVM opens the pass-through device, it compares the device's serial number to **check_sn**. If the number mismatches, AlphaVM rejects the device. This is a measure to avoid accidental corruption of a wrong device due to misconfiguration. The default value is empty, which means no check. Currently this option has effect only on FreeBSD.
- The options **trace_sense** enables logging of commands with sense data. Normally sense data is associated with errors or non-standard situations, so you may wish to enable it to see if something is going wrong. The default value is **no**.
- The options **trace_cmd** enables logging of all SCSI commands. This option is for debugging. The default value is **no**.
- The option **trace_io** enables logging of low-level IO operations. This option is for debugging. The default value is **no**.

3.8.1 Pass-through devices on Linux

On Linux the SCSI Pass Through devices are called /dev/sg*. Their mapping to real devices can be shown by means of the sg_map command. You may need to install the sg3-utils package.

A non-root user needs to set CAP_SYS_RAWIO capability to use this feature. The SG device should be accessible by the user. For instance: sudo chmod a+rw /dev/sg3. Otherwise the emulator fails with "Permission denied".

3.8.2 Pass-through devices on FreeBSD

On FreeBSD the SCSI Pass Through devices are called /dev/pass*. Use the command

```
camcontrol devlist -v
```

to see the available pass devices and their mapping to the real devices. Most controllers expose pass devices by default. Some device drivers only expose the devices they manage as pass devices if a sysctl is set. For the mfi controller, there are two ways to do this: "kldload mfip" will create pass devices for each PHYSICAL device attached to the controller. "sysctl hw.mfi.allow_cam_disk_passthrough" will create
a pass device for each LOGICAL device.

3.8.3 SCSI Tape

One of the typical uses of the SCSI Pass Through is to access a physical SCSI tape attached to the host.

The tape configuration is straightforward. For example it is :

```
scsi_unit tape0 {
      scsi_id = 6;
      device = ’/dev/sg3’;
}
```

3.8.4 iSCSI devices

The SCSI Pass Through can be used with devices available via iSCSI. The device must be available as /dev/sg*. This means that the Linux iSCSI initiator must be configured for the device.

iSCSI target can naturally be on the same machine or on a different machine.

Linux iSCSI target is compatible with the emulator. It can be directly used.

Windows 2012 SCSI target can require **suppress_vpd** and **emulate_mp1** to be enabled.

3.8.5 Non-SCSI disks

Linux converts SCSI commands to ATAPI device commands, in such a way that SCSI Pass Through can be used to access ATAPI devices. For instance, an ATAPI CDROM drive or a USB stick can be accessed this way.

Note that not all devices respond in a way accepted by OpenVMS or Tru64.

Some controllers supported by FreeBSD will convert SCSI commands to SATA, allowing non-SCSI disks to be used. This depends on the controller, and may require some of the options such as convert_cdb, suppress_vpd, and emulate_mp1.

Most SAS devices can be used directly, as they implement the SCSI command set but use a serial connection instead of parallel SCSI.

3.8.6 Booting from a SCSI Pass Through device.

Alpha can boot only from a device that supports the block size of 512 bytes. Most CDROM drivers have block size of 2048. However, Alpha CDROM drives are able to switch the logical block size to 512, which enables Alpha booting from these devices.

If your device is not capable of switching to logical block size of 512, AlphaVM will not be able to boot from it.

3.9 Serial port configuration

Serial port configuration section specifies how the port is connected. Currently the port is connected only to a virtual terminal. A virtual terminal can be connected to a terminal emulator. We advise a free terminal emulator PuTTY (written by Simon Tatham), which is widely used, although you can chose any other terminal emulator, although you can use another terminal emulator.

Configuration properties:

- The option **server** selects the way the serial line emulation is served. Currently there are two possible servers: **socket** and **serial**. The socket server maps the serial line to a TCPIP connection. Normally this connection has a terminal emulator (e.g. PutTTY) on the other side. The serial server maps the emulated serial line to a real host serial line (COM port). By default the value is **socket**. The **serial** server is available only in the professional version of the product.
- The option **device** specifies a host serial device (COM port). This device is used when the server is serial. This value is ignored when the socket server. Currently the SRM emulator ignores the SRM variable changes related to the serial line. By default it initializes the serial line to 9600 baud, 8 bits, No flow control, no modem control.
- The option **port** - is the TCPIP port number used to connect to the terminal emulator. The default value is 20000 for COM1 and 20001 for COM2. This value is ignored for the serial server.
- The option **logo** specifies whether the VM prints logo text on the terminal when then terminal is connected. The default value is **yes**. This can be disabled, which is useful in situations when the logo transmission breaks down the communication protocol.
- The option **session_log_enabled** controls whether the session log is enabled
- The option **session_log_append** indicates whether the session log is open in append mode.
- The option **session_log_binary** indicates whether the session log is open in binary mode.
- The option **session_log_file** specifies the session log file. The log file gets all the output in binary mode.

3.9.1 Connecting PuTTY

PuTTY has to be configured for using with the emulator serial lines. We advise to save this configuration under a descriptive name, so that you can reuse the configuration.

- Set connection mode to RAW.
- Set LOCAL ECHO to FORCED OFF
- Set LOCAL LINE EDITING to FORCED OFF

Example connections:

- localhost:20000 for COM1
- localhost:20001 for COM2

Note, that this section describes connection to emulated serial lines and not a usual network connection to the emulated machine via telnet or ssh.

3.9.2 Connecting a terminal via socat

You can connect your current terminal emulator to the console using **socat** command (install it if you do not have it):

```
socat -,raw,echo=0,escape=0x1c tcp:127.0.0.1:20000
```

The escape 0x1C is Ctrl-\; it allows you to escape from **socat**. This approach is contributed by Paul Sture to comp.os.vms.

3.10 Ethernet configuration

The AlphaVM system emulates Ethernet controller based on DEC21x4x also known as Tulip.

Some Alpha systems have one or two built-in Ethernet adapters. Configurations eth0 and eth1 correspond to the built-in adapters in this case. We recommend specifying configurations of such built-in adapters explicitly, even if they are not used.

The emulator communicates with the real Ethernet using libpcap. The user has to provide the information about the connection. In particular, the user has to specify which Linux network interface will be used by the emulator.

Configuration properties:

- The option **type** specifies the type of the emulated Tulip adapter:
  - dec21040 – a 10Mbit controller also known as DE435
  - dec21143 – a 100Mbit controller also known as DE500
- The option **server** defines how the network is served by the emulator. The allowed options are **dummy, pcap** and **tap**.
  - Dummy basically means that the wire is unconnected.
  - Pcap uses the libpcap functionality to access the real network.
  - Tap is used with tap network interfaces.
- The option **mac_mode** specifies how the emulated station MAC address is constructed.
  - **user** – the emulator will use the **mac_address** as the station address. Make sure all MAC addresses are unique on your network. This is the default value.
  - **host** – the emulator will use the MAC address of the host NIC. This setting is to be used NICs dedicated only to AlphaVM. The Windows IP protocols have to be disabled on this NIC, otherwise a MAC address conflict would occur. Two systems would have and IP stacks using the same MAC address: the guest and the host systems.
  - **auto** – the address is automatically generated. This option is convenient, because you do not have to invent a unique address. Note however, that the address will not be unique when two instances of the emulator process use the same host NIC. This is because the instances do not coordinate the MAC address allocation. Use the user defined address mode if you are configuring several instances sharing the same host NIC. Within one instance, if several AlphaVM NICs use the same host NIC, the addresses will be unique.
- The option **mac_address** specifies a MAC address to be used when **mac_mode** is **user.** If **mac_mode** is not **user**, this field is ignored. The address is specified in a TCPDUMP format, as a hexadecimal number. The default address is 0x08002B000001, which is 08:00:2B:00:00:01. If there are several emulators on your network, make sure their Ethernet controllers have a unique MAC addresses. This is to avoid MAC address conflicts.
- The option **interface** specifies a Linux/FreeBSD network interface used to connect to the network. You can list the available host NICs using “ifconfig –a”. Linux examples: eth0, eth1. FreeBSD examples: bge0, bge1, de0, de1.

- The option **slot** specifies the PCI slot to which the adapter is plugged. The default behavior is automatic; thus, you do not have to specify this option or know about slots. The automatic behavior covers most cases with OpenVMS. Tru64, however, is very sensitive to changes in the hardware configuration. When you copy your disk images from the real system, it can be required to specify the slots in such a way to reflect the configuration of the real. The number of slots depends on the actual emulated system. The mapping of the slots to PCI hoses and IDSELs also depends on the system.
- The option **rx_buf_size** specifies the pcap RX buffer size in megabytes. Zero means that the default PCAP buffer size will be used.
- The option **tx_buf_size** specifies the pcap TX buffer size in megabytes. This option is currently ignored; it exists for compatibility with Windows where it specifies the TX queue size.
- The option **dma_cache** specifies whether the NIC (Tulip) caches the DMA translations. This option allows to significantly decrease the number of DMA translations. It works if the DMA rings reside at a constant DMA addresses, which appears to be true for the supported guest OSes. The default value is false.
- The option **server_filtering** specifies whether a PCAP level packet filter is used (BPF). It is yes by default. The option can be used to disable the filtering to deliver all packets to AlphaVM.
- The option **stat_period** specifies a period in seconds used to log the statistics counters for pcap.
- The option **trace_rx** enables the tracing of packets received by the NIC.
- The option **trace_tx** enables the tracing of packets transmitted by the NIC.
- The option **trace_filter** enables tracing of filter changes on the level of the NIC and PCAP.

The emulator can share the same host network interface with other programs running on the host. However, the emulator maintains a different Ethernet address from the host Linux system. It is necessary that the address is different. It guarantees that packets meant for the emulator are not mixed with packets meant for the hosting OS.

For performance reasons you may wish to use a dedicated network interface for the emulator. To achieve this, disable all Linux protocols on the dedicated NIC. In this case Linux will not interfere with the activity of the emulator. You may also wish to use the same Ethernet address as the real address of the dedicated host NIC.

3.10.1 Communication between the host and AlphaVM

When both the host and the emulator use the same network interface, there is a problem of communication between the host and the guest. This option works only for communication with a remote system. However often it is desired to communicate between the host and the guest. For instance, you may wish an X-server running on the host to connect to the guest. This section describes how to configure network to allow for such communication.

The simplest solution is to use a dedicated host network interface for the emulator. Thus, you should have two network interfaces in your host: one used by the host and one by the emulator. They should both be connected to the same network. In this way packet sent between the host and the emulator go through the real network. It works just like it normally works with a remote machine.

Another solution is available only on Linux. On FreeBSD it is not yet implemented. It involves a virtual network within your system to communicate between the host and the emulator. A virtual network interface is bridged with the real network by means of a virtual bridge. The solution is described in the next section.

3.10.2 Configuring RHEL Ethernet interface

On RHEL some host NIC features must be disabled in order for AlphaVM to work via PCAP. Some NIC offloads can coalesce packets and make them look corrupted to AlphaVM. For the network interfaces used by AlphaVM the following line has to be added to the end of the NIC configuration file (for instance /etc/sysconfig/network-scripts/ifcfg-eth1) :

```
ETHTOOL_OPTS="-K ${DEVICE} tso off ufo off lro off gso off gro off"
```

The whole configuration file will look as follows (CentOS 6.10):

```
$ cat /etc/sysconfig/network-scripts/ifcfg-eth1

DEVICE=eth1

TYPE=Ethernet

UUID=3b240b30-1459-4141-8a32-cdaccebe2b99

ONBOOT=yes

BOOTPROTO=none

NM_CONTROLLED=no

HWADDR=66:71:EF:62:46:19

IPV6INIT=no

NAME="System eth1"

USERCTL=yes

ETHTOOL_OPTS="-K ${DEVICE} tso off ufo off lro off gso off gro off"
```

Note that this interface is configured for dedicated usage by AlphaVM: NM_CONTROLLER=none.

3.10.3 Configuring Debian Ethernet interface

Make sure ETHTOOL is installed:

```
sudo apt-get install ethtool
```

Edit /etc/network/interfaces for the dedicated AlphaVM network interface(s) to contain required ethtool options. For instance:

```
auto ens19
```

```
iface ens19 inet manual

        pre-up ifconfig $IFACE up

        post-down ifconfig $IFACE down

        post-up ethtool -K $IFACE tso off ufo off lro off gso off gro off
```

3.10.4 Using TAP network interface on Linux

A TAP network interface can be used with AlphaVM. Usually it is used in combination with a network bridge that bridges TAP interface(s) with the real network interface. Modern UNIX based systems enable TAP configuration in various ways.

The solution described below is designed by the authors of the SimH emulator. The example is for Debian.

```
# install required packages (do it once)

apt-get install make

apt-get install libpcap-dev

apt-get install bridge-utils

apt-get install uml-utilities

# Do it as root or prefix the commands with sudo

# create a TAP interface for a user

tunctl -t tap0 -u artem

ifconfig tap0 up

# Now convert eth0 to a bridge and bridge it with the TAP interface

brctl addbr br0

brctl addif br0 eth0

brctl setfd br0 0

ifconfig eth0 0.0.0.0

ifconfig br0 192.168.1.2 netmask 255.255.255.0 broadcast 192.168.1.255 up

# set the default route to the br0 interface

route add -net 0.0.0.0/0 gw 192.168.1.1

# bridge in the tap device
```

```
brctl addif br0 tap0

ifconfig tap0 0.0.0.0
```

Italic denotes values that you have to replace according to your configuration. Use the ifconfig command without arguments to obtain the resulting configuration. The result should look like shown below.

```
br0       Link encap:Ethernet  HWaddr 00:18:8b:cb:a8:8f
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::218:8bff:fecb:a88f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3780 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1863 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:595198 (581.2 KiB)  TX bytes:194247 (189.6 KiB)

eth0      Link encap:Ethernet  HWaddr 00:18:8b:cb:a8:8f
          inet6 addr: fe80::218:8bff:fecb:a88f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2201 errors:0 dropped:0 overruns:0 frame:0
          TX packets:357 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:503824 (492.0 KiB)  TX bytes:52992 (51.7 KiB)
          Interrupt:18

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:13373 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13373 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

```
          RX bytes:734785 (717.5 KiB)  TX bytes:734785 (717.5 KiB)


tap0      Link encap:Ethernet  HWaddr 1a:c7:38:fa:e6:22
          inet6 addr: fe80::18c7:38ff:fefa:e622/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1634 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3542 errors:0 dropped:84 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:159566 (155.8 KiB)  TX bytes:620402 (605.8 KiB)
```

The emulator must be configured to use the server **tap** and the interface **tap0**.

Please make sure that the emulator uses a MAC-address different form that of tap0.

Note that the configuration will disappear after a reboot. Put it in a startup script to configure it automatically.

The host network parameters can be automatically determined as follows:

```
HIP=`/sbin/ifconfig eth0 | grep "inet addr" | gawk -- '{ print $2 }' | gawk -F : -- '{ print $2 }'`
HMASK=`/sbin/ifconfig eth0 | grep "inet addr" | gawk -- '{ print $4 }' | gawk -F : -- '{ print $2 }'`
HBCAST=`/sbin/ifconfig eth0 | grep "inet addr" | gawk -- '{ print $3 }' | gawk -F : -- '{ print $2 }'`
HGATEWAY =`/sbin/route -n | grep ^0.0.0.0 | gawk -- '{ print $2 }'`
```

These variables can be used in a script to create the whole setup from the eth0 configuration:

3.11 VM launching configuration

Configuration properties:

- The options **min_working_set** and **max_working_set** specifies the minimal and maximal working set limits. These are advanced settings. Do not change them unless you are sure what you are doing. Wrong settings can badly impact the emulator and the system performance. The default value is zero, which means that the virtual machine sets the limits automatically. Working set limits can be changed to tune the VM performance in case the system defaults do not work well. Working set is the amount of physical memory used by the process, in our case the VM. Too low working set limit can cause excessive page faults in the VM on the emulated memory access, which can disturb timing of the emulated CPU. Too high working set limits can lead to lack of resources for the host system, which degrades the whole system performance including the VM. The option **min_working_set** is currently ignored on Linux.

- The option **process_affinity** specifies a CPU affinity mask to be used by the VM process. Each CPU in the mask specifies whether the VM can run on the corresponding host CPU. This feature allows limiting the amount of the CPU resources used by the VM. The default value is zero, which means that the VM can run on any available CPU.

3.12 VM Logging configuration

The virtual machine produces log to standard error.

Configuration properties:

- The option **file** is currently unused. The logging is sent to standard error.
- The option specifies whether the log file is appended or truncated on every run. The default value is false (truncate every time). Note that in append mode the file can become huge over time. Note also that when you get a problem with the emulator, you should save the log file before restarting of the emulation process, otherwise the log of the erroneous run will be lost. We recommend to use non-append mode in conjunction with non-zero **max_backups** to save the log files from the previous VM runs.
- The option **max_backups** specifies the number of of file backups maintained by the virtual machine. The backups have the form of <logfile>.<version>. Newer versions have higher version numbers. For example, when the log file is specified as /AlphaVM/Test/vm.log, the emulator will create 3 backups: /AlphaVM/Test/vm.log.1, /AlphaVM/Test/vm.log.2, /AlphaVM/Test/vm.log.3. The default value is 3. The emulator creates a backup each time the log is opened in non-append mode. If the maximal size of the log file is specified, the old log is saved as a backup and the new log is opened.
- The option **max_size** specifies the maximal size of the log file in megabytes. When the size is reached, the log file is closed and the new log file is created. If MaxBackups is not zero, the old log file will be saved as a backup. Essentially this logging method creates a ring of log files. This method ensures that the logging on the server would never exceed the size of the log file and its backups.
- The option **time_mode** specifies the time logging mode. By default the local time stamp is printed.
    - **none** - no time stamp
    - **counter** microsecond tick counter is printed. This value can be used when a lot of tracing is produced to minimize the time needed to obtain the timestamp.
    - **utc** – log UTC timestamp. The actual time format is specified by **TimeFormat**.
    - **local** – log local time stamp. The actual time format is specified by **TimeFormat**.
- The option **time_format** specifies the format of the timestamp for UTC and local modes. It is a single quoted string. The format specification is the same as for strftime() function. The default value is '%Y-%m-%d %H:%M:%S', which prints time like 2014-05-16 17:47:46.
- The option **time_fraction** specifies whether to log time fraction. Time fraction is appended in microseconds to the time stamp in the format .NNN. The option is Boolean. The default is yes.

3.13 Licensing configuration

Configuration properties:

- The option **host** is the IP address of the system running the EmuVM licensing service. When using a USB dongle, this is normally *localhost*. For evaluation set the evaluation server IP provided by EmuVM.
- The option **port** is a number used to connect to the licensing service. Use 19991 with the evaluation license server. Use 19992 with a USB dongle server.
- The option **username** is a username used to connect to the licensing service. For evaluation, use the user name provided by EmuVM. When using a USB dongle it is usually sys0 unless another name is provided by EmuVM.
- The option **password** is a password used to authenticate the user at the licensing service. When using a USB dongle, please use *default*. Otherwise use the evaluation password provided by EmuVM.

3.13.1 Configuring for evaluation

The AlphaVM evaluation can be done using a remote EmuVM server. You will receive the server IP address, port number, username and password to be used. The EmuVM evaluation server uses the port 19991.

Please make sure the outgoing port is open at your firewall and anti-virus software. Please first use ping to check the availability of the server.

3.13.2 Configuring with USB dongle

The dongle service is called keylok_service. It is available in the distribution package. Please run it before starting the emulator. It is advised to make it start automatically at the host system startup.

Normally AlphaVM is protected by a physical USB dongle plugged into the host system. In this case a typical configuration is host='localhost', port=19992, user='sys0', password='default'.

Sometimes it is convenient to have the dongle in another system. Start the service on that machine and configure the host setting appropriately.

3.14 Configuration of multiple instances

It is possible to run several instances of the emulator on a single host system.

The following steps are needed to configure several instances

- Make a separate directory for each system configuration.
- Place the configuration file for each system in the corresponding directory.
- Place the private disk and tape images in the corresponding directory.

- Set unique MAC addresses for each network card in each configuration.
- Make sure the serial line configuration use unique ports.
- Set CPU affinities in such a way that different instances use different host CPUs. CPU affinity is a bit mask where each bit represents one host CPU core or hyper-thread. When a bit is 1, the corresponding core is used to run the corresponding instance of AlphaVM. The table below contains the affinity setting examples.
- For AlphaVM-Pro: set the licensing information for each instance. The information about license settings will be provided when you purchase the product.

Examples of setting CPU affinities:

Host cores or hyper-threads	Example of the host system	Number of AlphaVM instances	Max Alpha CPUs	Affinities per instance
4	I5 with hyper-threading on, or I7 with hyper-threading off	2	1	0x0000000000000003 0x000000000000000C
8	I7 with hyper-threading on	2	2	0x000000000000000F 0x00000000000000F0
8	I7 with hyper-threading on	4	1	0x0000000000000003 0x000000000000000C 0x0000000000000030 0x00000000000000C0
6	I7-3970K with hyper-threading off	2	2	0x0000000000000007 0x0000000000000038
6	I7-3970K with hyper-threading off	3	1	0x0000000000000003 0x000000000000000C 0x0000000000000030
12	I7-3970K with hyper-threading on	3	2	0x000000000000000F 0x00000000000000F0 0x0000000000000F00
12	I7-3970K with hyper-threading on	2	3	0x000000000000003F 0x0000000000000FC0
12	I7-3970K with hyper-threading on	6	1	0x0000000000000003 0x000000000000000C 0x0000000000000030 0x00000000000000C0 0x0000000000000300 0x0000000000000C00

4 Emulator operation

4.1 Starting the emulation

The emulator is started from the command line as follows:

```
alphavm_pro myconfig.emu
```

4.2 Stopping the emulation

Please do not stop the emulator by means of *kill* or ^-C unless it is really necessary. This corresponds to an abnormal system power failure and can cause troubles with the guest operating system or other guest software currently running in the emulator. Instead, shutdown the guest system and use SRM **power** command to "power down" the emulated system, which causes the VM process to exit normally.

5 Migration

A real system can be replaced by the emulator software. Firstly, the emulator should be configured to reflect the real system as close as possible. Secondly the software should be transferred to the emulator.

5.1 Migration by copying disks

The simplest way of migration is by copying the real system disks to disk images and then using these disk images to run the emulation. Thus, the whole OS, software and data are copied. The new system behaves in the same way as the old one.

Unfortunately, this method does not always work. At the moment we cannot emulate all kinds of Alpha systems and all kinds of peripheral devices. Some OSs and applications are flexible and can run on a different hardware configuration without changes. Others require more or less complicated reconfiguration.

5.1.1 Copying disks using a Live Linux CD

When a system is booted from a disk, this disk cannot be modified while being copied. To avoid such problems you can boot a Linux system form so called Live CD (for instance, Gentoo LiveCD). In this case you get a booted Linux system that does not use any of you OpenVMS or Tru64 disks. This Linux system can be used to copy your disks by means of the dd command. You will have to use a storage on the network to store the resulting disk images.

When copying disks please use the whole disk devices like /dev/sda, rather than partition devices (like /dev/sda1)

Please note that Tru64 can be very sensitive to configuration changes. When the copied disk image is booted in the emulator it may fail due to difference in the configuration. In this case you can boot the generic kernel and reconfigure/rebuild your kernel as usually.

See a detailed example on emuvm.com/migrate_vms.php.

5.1.2 Migration of OpenVMS using backup /image

The usual way to migrate OpenVMS is to use backup /image copies of the disks. The following command can be used to make such backups:

```
$ mount dka100: /over=id

$ backup /image/verify dka100: dka500:[000000]foo.sav/sav
```

You can use the command to backup your system disk.

The following commands can be used to restore disk images:

```
mount dka100: /foreign

backup /image/verify dka500:[000000]foo.sav/sav dka100:
```

The migration process can be outlined as follows:

1. Make backup /image backups of all disks in your system and make them available via the network.
2. Install AlphaVM and create N+1 empty disk images. Add the disk images to the emulated system. One disk will be used for a freshly installed copy of OpenVMS. The other disks will be used to restore your original disks. The AlphaVM disk images can be larger than the original disks.
3. Install a fresh copy of OpenVMS on AlphaVM and configure the network to access the disk backups.
4. Restore the disk backups from the files to the empty emulated disks.
5. Boot from the restored system disk.

It is advised that your fresh system disk is made large enough to contain any of the *.sav files (or all of them). It this case you can restore from a local copy rather than from a DECNET remote copy. It is possible to use TCPIP to ftp files and you do need to configure DECNET.

5.1.3 Copying disks on OpenVMS

On OpenVMS a proper disk image can be created by means of backup /physical. Please note that the physical image has the same size as the original disk. Therefore, you need to have enough storage to store the resulting file. Please note that you can use a network path for the destination file.

5.1.4 Copying disks on Tru64

On Tru64 a proper disk image can be created by means of the dd command. Please note that the physical image has the same size as the original disk. Therefore, you need to have enough storage to store the resulting file. Please note that you can use a storage available via the network as the result, for instance NFS.

5.2 Migration by reinstalling software from scratch

When it is impossible or inconvenient to copy disks, the software can be installed on the emulator in the usual way. These are the steps to be done:

- Install the OS and its layered products.
- Install and configure the application software.
- You may need to copy data from the old system.

6 Getting started

6.1 Getting started on Linux

The goal of this section is to provide simple step-by-step instructions to start with the emulator.

You need to do the following steps to configure and run the emulator. The procedure is tested under a Debian 6.x distribution. It should be similar on other Linux distributions.

1. If you plan to use the emulator with Ethernet under a non-root user, you need to set capabilities CAP_NET_RAW and CAP_NET_ADMIN. You may wish to skip this step if you plan to run under root. For instance, run the following commands under root:

```
apt-get install libcap2-bin

setcap cap_net_raw,cap_net_admin=eip alphavm_pro
```

   To check that they are added

```
getcap alphavm_pro
```

   The result should look like:

```
alphavm_pro = cap_net_admin,cap_net_raw+eip
```

   You may wish to skip this step if you plan to run under root.

2. Install PUTTY. It is a terminal emulator. Use a command like:

```
apt-get install putty
```

   In principle you can use another terminal emulator, but we recommend putty, because it has all the necessary options and has proven to work well with the emulator.

3. Create empty disk images (one or more). For instance, run

```
./mkdisk rz59 mydisk.dd
```

This would create a disk image of RZ59, which was 8.5 GB. The disk image will be empty and it will be saved into the file mydisk.dd. For other options run

```
./mkdisk
```

It will give the usage information and the disk table:

```
Create empty disk image

USAGE: ./mkdisk <type> <file>

<type> is the disk type

rz26      - DEC RZ26, 1Gb

rz28d     - DEC RZ28D, 2Gb

rz29b     - DEC RZ29B, 4Gb

rz59      - DEC RZ59, 8.5Gb

hdd10gb, hdd20gb, ... hdd50gb - EmuVM image of a specific size
```

4. Copy and edit the configuration file example.emu.

```
cp example.emu mysystem.emu
```

There are the following places you may need to change:

- Set the system type - it is es40, but you can choose ds20, or ds10.
- The number of CPUs. The number of CPUs depends on the emulated system type. It is maximum 4 for the emulated chipset.
- Memory size.
- The CPU server can be set to the fastest available with your license (e.g. jit3).
- COM1/COM2 - leave them as is. Although, you may want to change the port numbers.
- scsi_disk dka0 - Here you will need to specify your disk image file instead of the one specified in the example. Thus, set

  file='/full_path_to_my_disks/mydisk.dd';

- If you need more disks copy the section

  scsi_disk dka0 { ... }

  to

  scsi_disk dka100 { ... }

  and change the scsi_id to 1. Be sure that the SCSI paths are unique. Also set another image file.

- The other disk parameters are not important, just ignore them for now.
- in scsi_cdrom iso { ... } set the path to the iso image that contain the OS you want to install. The SCSI ID is set to 4.

- With ethernet you need to specify server=pcap and to specify the Linux network interface you want the emulator to use, usually eth0 or eth1. The server=dummy disables the network interface.
- Specify a unique mac_address. If you just have one emulator on your network, you do not need to change this setting.

5. Run the emulator:

```
alphavm_pro mysystem.emu
```

6. Connect terminal emulator to the console. Run putty and configure the AlphaVM console session with the following parameters:
   - Use localhost:20000 (you can also connect remotely)
   - Use RAW mode (not TELNET or SSH)
   - Set LOCAL ECHO to FORCED OFF
   - Set LOCAL LINE EDITING to FORCED OFF
   - Do not forget to save the session under a name like AlphaVM_COM1
   - Connect putty to the emulator
   - You should see the emulator logo on the terminal screen
   - You can also connect a terminal to AlphaVM COM2 (port 20001).
7. Boot the emulator. Once you see the logo on the terminal you can press ENTER several times to see SRM boot prompt ">>>". When you see it you can type the usual commands to boot the emulator.
   - Use BOOT DKA400 to boot from you installation CDROM and install the OS.
   - Use BOOT DKA0 to boot from the hard disk with SCSI ID 0. (The system must be installed on this disk before you can boot form it)
   - Use SH DEV to see the configured devices.