OpenVMS Technical Journal V14

SimH Supports Availability Manager Development Environment at HP

Barry Kierstein, Software Engineer

Introduction

During my tenure at Hewlett-Packard, I served as the project leader for ***Availability Manager*** on OpenVMS. As such, I was responsible for its development and testing on the VAX, Alpha, and Integrity platforms. This article describes the deployment of the open source SimH VAX emulator to replace aged VAX hardware while maintaining the *Availability Manager* test environment. This article was sponsored by Migration Specialties.

The article is in two main sections. The first section tells the story of why SimH VAX was selected and how it was utilized to solve legacy hardware and testing issues. The second section, Configuration and Implementation, provides details on how the SimH VAX environment was created.

The Problem: Legacy VAX Test Hardware

Over time, the primary development and test cluster for *Availability Manager* evolved to consist of a number of VAX, Alpha, and Integrity systems. These machines were good solid machines, but some were getting to be pretty old, especially the VAX systems. By 2007, I was concerned that the VAX systems would eventually fail. Since I had a limited hardware budget, this would leave me in the difficult position of supporting VAX systems in the field without any VAX hardware in the lab. Much of the *Availability Manager* development work involved device drivers, with the associated potential of crashing systems during development and testing. Hence, using machines outside the *Availability Manager* development cluster was not an option.

The Virtual Option

In the summer of 2007, VAX hardware reliability concerns prompted me to research replacement options. I needed four VAX systems to replace the VAX workstations that I possessed at that time – one for each version of OpenVMS supported by *Availability Manager* (OpenVMS V6.3, V7.1, V7.2, and V7.3). I was looking for something that would fit my budget and work on the available Windows PC systems.

Virtual machines were becoming common place and the idea of a virtual VAX was intriguing. Virtual machines are on ongoing interest for me. I had studied the Virtual Machine (VM/CMS) operating system by IBM in college, seen commercial VAX emulators demonstrated at various forums, attended VMware sessions at the HP Technology Forum, and used the Microsoft Virtual PC software.

Virtual VAX Advantages

A virtual VAX offered the following advantages for *Availability Manager* development and testing:

- Easy to backup and restore a virtual machine
    - Allowed for quicker recovery from development bugs
    - Allowed a quick system restore to a known state for repeated tests
- Easy system reconfiguration of memory, disk drives, and network adapters
    - Allowed quick move of a disk drive from one system to another
    - Allowed development and test system configurations that I could not previously afford using physical hardware, such as 512MB of memory, many disk drives, and multiple network adapters

Evaluating SimH

After investigating alternatives, the free, open source SimH VAX emulator looked promising. The SimH website, http://simh.trailing-edge.com/, provided the then current installation kit, version 3.7-0. I downloaded the software and started reading the documentation. I also subscribed to the SimH mailing list (the address is under "Help With SimH" on the SimH homepage). Between the documentation and some postings from the SimH website, I was able to get my first virtual VAX up and running.

Working with this simulation, I learned a number of things:

- The *Availability Manager* successfully installed and ran in the virtual environment.
- The ability to backup and restore virtual machines was as easy as expected, opening up possibilities of repeated testing of a system starting from a known state.
- The ability to reconfigure the virtual machines by changing the amount of memory, changing the number of disk devices, etc. was as easy as expected, enhancing configuration testing capabilities.
- The ability to transfer a disk from one virtual machine to another was as easy as expected, which improved operationally efficiency and flexibility.
- Each virtual machine needed its own network adapter and MAC address.

Handling Multiple Network Adapters

The need for separate network adapters was based on a subtle, but important behavior. This involves how applications sharing an Ethernet-based network adapter are able to view traffic from each other. The adapter, by default, does not send the outbound traffic from one application to the other applications on a shared adapter. In my case, if two virtual machines shared the same adapter, outbound traffic from one of them was not seen by the other. So, if the two virtual machines were part of the same cluster, they wouldn't see the cluster traffic from each other. As a result, the two virtual machines cannot form a cluster connection between them.

This network adapter behavior is well known and can be addressed in a couple of ways. One solution is using one network adapter per virtual machine. This is the simplest setup and the one with better performance. The other is to install additional software that allows applications sharing a network adapter to see the outbound traffic from each other. This solution allows one to use fewer physical network adapters.

In my case, I wanted to keep the software setup as simple as possible so I obtained enough network adapters to have one per virtual machine.

SimH Implementation

Evaluation of the initial SimH virtual VAX established that the emulator would support the OpenVMS VAX environment required for *Availability Manager* development and testing. I had a single dual-CPU PC available to host four instances of the VAX emulator. The limited host hardware impacted performance of each virtual VAX, but since I didn't need great performance, this was acceptable. I used the following steps to generate the initial virtual VAX configurations:

- Installed the additional network adapters into the host system.
- Built the configuration files for each virtual VAX system.
- Assigned a unique MAC addresses to each system.
- Created virtual disk drives for each VAX system. The virtual drives appear as large files on the host system and are referred to as container files. Virtual drives can be created using the SimH ATTACH command.
- Test booted each virtual VAX to the boot prompt and verified the configuration via console commands.

The next step was to migrate the drive contents from the four legacy VAX systems to the virtual VAXen. I accomplished this by installing OpenVMS on one virtual VAX. I then configured this virtual machine to use all of the virtual machine disk drives, booted the system into the cluster, and used $ BACKUP/IMAGE to move the drive contents from the physical VAX systems to the virtual VAX disk drives. Using image backups on live disks does entail the risk of not picking up all the contents of open files. However, the open files on my systems were mostly log files so this was not an issue.

Once I had migrated the drive contents, I set the four virtual VAX to their final configuration and booted them. They joined the *Availability Manager* development cluster and acted like their VAX hardware counterparts!

After finishing the initial conversion of the four VAX systems, I created four more virtual systems. These had the same configuration as the first four systems, but with a fresh install of OpenVMS. These virtual machines were for testing initial installations of *Availability Manager* kits. I had arranged for some disk space from the main OpenVMS cluster for the virtual machine backups. I copied the virtual machine container files to the OpenVMS cluster to establish a base copy of the virtual machines before installing *Availability Manager*. With this backup, I could install on these systems and then simply copy the container files from the OpenVMS cluster to restore the systems to their initial state. Much faster than using tapes, etc. on physical systems!

Configuration and implementation details are in the section "Configuration and Implementation" at the end of this article.

Putting SimH to Work

Development and testing of each version of *Availability Manager* requires a fair number of test installations. A great bonus of the virtual VAX setup was the ease in which a virtual machine could be restored to a known state after a test. By copying the disk container files from the OpenVMS cluster to the PC, I didn't have to fiddle with standalone backups, tapes, etc. I was able to do a thorough job of testing than before. Part of this expanded testing involved changing the amount of memory each virtual machine had, the numbers of disk drives available, etc. This type of testing had been limited in the past by physical hardware availability.

Another virtualization benefit was that I could do more radical testing since I didn't have to be as careful with the virtual machines as I did with the original VAX hardware. Unlike physical hardware, recovery from a bad software install or test could be achieved by simply restoring copies of the base system disk container files from the main OpenVMS cluster.

While performance of each virtual VAX was a bit slow due to the limitations of the host system, it was acceptable. Idle mode detection was available in the version of SimH I deployed which helped alleviate performance issues.

As it turns out, my premonition about the physical VAX hardware came to pass. In the winter of 2008, there were a number of power interruptions and outages that did in the VAX hardware. It was sad to see the old hardware go, but fortunately, the applications they supported lived on in their new virtual home.

Conclusion

The time taken to evaluation and implement the SimH VAX emulators proved to be a good investment. System reliability, test efficiency, test flexibility, and overall productivity improved after installing the SimH VAX systems. This in turn allowed me to do additional development and testing. *Availability Manager* is a better product as a result.

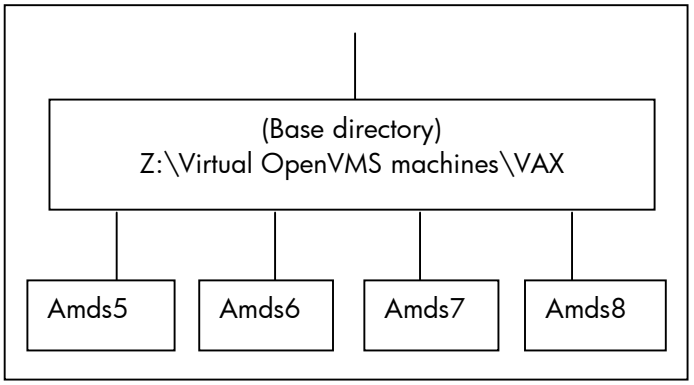
Configuration and Implementation

This section covers the configuration and implementation details of the *Availability Manager* SimH test environment I created.

Configuration for SimH is not complex. The main areas of focus are disk volumes, network adapters and the console for the virtual machine.

A SimH virtual machine consists of the disk volume container files, a file to contain the non-volatile RAM and an editable text file with the configuration details. I created the following directory structure to contain the files for all the virtual machines.

Directories Amds5 through Amds8 were named after the OpenVMS VAX systems being emulated. These directories housed the configuration and virtual disk files unique to each emulated VAX system. The configuration file for each system specified the non-volatile RAM and disk container files with relative path names from the base directory. This made moving the virtual machines from one place to another easier.



Specifying a configuration file for SimH VAX is fairly simple – simply specify the path and file name as the first parameter. Running SimH on Windows, I created a shortcut and set the Target field to the following:

```
"Z:\Virtual OpenVMS machines\VAX\vax.exe" amds8\simh_amds8_config.txt
```

Figure 1: SimH shortcut.

I also set the Start in field to "Z:\Virtual OpenVMS machines\VAX", the base directory, for the relative path references in the configuration file.

This is the configuration file for one of the virtual machines – AMDS8. The configuration defines two disk volumes and one network adapter. It also has a number of commented-out commands as a reminder of how to configure various items. Note that this configuration file was used with SimH V3.7-0.

```
; Load CPU microcode
;
load -r ka655x.bin
;
; Attach non-volatile RAM to a file (default boot device settings,
etc.)
;
attach NVR amds8\ka655.nvr
;
; Set the memory to 128MB
;
set cpu 128m
;
; Set the CPU to allow the emulator to idle when OpenVMS is idle.
;
set cpu idle
d idle_ipl 8
d idle_wait 1000
;
; Set the halt to return to the ROM console
;
set cpu conhalt
;
; Set the localhost TELNET port for the console
;
set TELNET 5308
;
; Enable and disable various devices
;
set RL disable
set LPT disable
set TQ disable
set TS disable
set rq0 ra92
attach rq0 amds8\amds8_sys.dsk
set rq1 ra92
attach rq1 amds8\amds8_user.dsk
set rq2 disable
set rq3 disable
;
; Set MAC address and adapter
;
set xq mac=00-08-C8-08-CE-DE
attach xq eth4
;b cpu
```

Figure 2: Sample SimH VAX Configuration File.

I put a number of comments in this file as well as various commands that are commented out. I left these in as a reminder of what possible. Here are some highlighted sections:

```
; Set the memory to 128MB
;
set cpu 128m
```

Figure 3: Setting emulator memory parameter.

This command sets the physical memory for the virtual machine. For the MicroVAX 3900 emulator, this has been extended so that it can emulate up to 512MB from the usual maximum of 64MB.

```
; Set the localhost TELNET port for the console
;
set TELNET 5308
```

Figure 4: Setting the console port.

This command defines the console input/output to port 5308. Note that the equivalent command for SimH 3.8-1 is "set CONSOLE TELNET=5308". Since I was running a number of the emulators at the same time, I had a system of starting the port numbers at 5300 and then appending the node ID number (8 for Amds8, etc.).

Using the telnet port, the virtual machine console can be connected to the terminal emulator of your choice. I decided to use the freeware PuTTY emulator[1] since I was familiar with it and satisfied with its VT emulation.

I created a PuTTY session – AMDS8 Console – that specified the Telnet port. Then, I created a shortcut for the Amds8 console that started PuTTY and specified the PuTTY session. The Target field of the shortcut is as follows:

```
"R:\Program Files\PuTTY\putty.exe" -load "AMDS8 console"
```

Figure 5: PuTTY shortcut.

To start Amds8, I would first double-click on the Amds8 SimH shortcut to start the emulator. The emulator would start to execute in a Windows console window and then prompt for input with a `sim>` prompt. At this point I would double-click on the Amds8 console shortcut and PuTTY would connect to the emulator console. I would then enter the `B CPU` command at the `sim>` prompt to start the virtual machine. The console displayed the normal output for a MicroVAX 3900, and I interacted with this console as if it were the real thing!

```
set rq0 ra92
attach rq0 amds8\amds8_sys.dsk
set rq1 ra92
attach rq1 amds8\amds8_user.dsk
set rq2 disable
set rq3 disable
```

Figure 6: Virtual disk setup.

These are the commands used to set up the two virtual disk drives for the virtual VAX. By default, there are four RQ devices per controller. I needed only two RQ devices so, I explicitly disabled the other two. If these devices aren't explicitly disabled, they show up under OpenVMS as two online disk drives. However, when you try to mount them, OPCOM complains that the medium is offline and that you must mount a device for the drive – confusion that is easily avoided by taking advantage of the disable feature.

If a virtual disk needs to be moved from another machine to this one, all that is needed is to set RQ2 or RQ3 to specify the location of the container file, and the disk is available to the virtual machine!

There are a number of other disk controllers that can be enabled, allowing a virtual machine to emulate a fairly large number of disk drives. In SimH version 3.8-1, the VAX 11/780 emulator can emulate up to eight RP04/05/06/07 or RM02/03/05/08 disks, four RL11/RL01/02 disks, and 16 UDA50 MSCP disks on four controllers, as well as TS11, TUK50, and TM03 tape drives and two DEUNA/DELUA Ethernet controllers. The MicroVAX 3900 can emulate up to 512MB of memory and has a similar number of disk, tape, and Ethernet controllers as the VAX 780 emulator.

[1] PuTTY is available at http://www.chiark.greenend.org.uk/~sgtatham/putty/.

```
set xq mac=00-08-C8-08-CE-DE
attach xq eth4
```

Figure 7: Setting up a network adapter.

These are the commands used to set up the network adapter. I set the MAC address to what I found was a unique address in the network segment. Setting the MAC address is an option; it is not required. Since I was hosting four SimH VAX emulators on one physical system, I wanted to ensure each emulated VAX had a unique MAC. Note that eth4 is the mnemonic for the fourth adapter on the host machine (Amds5 took the first one adapter on my system, Amds6 the second, etc.).

Eth1 is bound to the first adapter, Eth2 is bound to the second adapter (if it exists), etc. If you type in the "attach" command interactively at the SimH prompt, SimH displays which network adapter is bound to Eth1, Eth2, etc. If a binding doesn't exist for a particular mnemonic, an error message is displayed.

Once this configuration was done, the virtual machines were ready to run.

Installing OpenVMS

Installing OpenVMS onto a newly created virtual machine can be done through a CDROM device on the virtual machine. An example of configuration for the CDROM drive is shown below:

```
set rq2 cdrom
attach –r rq2 VAXVMS073.ISO
```

Figure 8: Setting up a CDROM drive.

Note that the contents of the OpenVMS installation CD have been transferred to an ISO file on the host system. There are a number of Windows applications that can create an ISO file from a CD, and many Windows systems have an application that does this already installed. Once the ISO file has been created, then it can be used by the virtual machine for the OpenVMS installation.

About the Author

Barry Kierstein has worked with OpenVMS since 1981, and worked for Hewlett-Packard for almost fourteen years. His work at Hewlett-Packard was with the System Management group within the OpenVMS organization and is most known for his project leadership work and advocacy of the *Availability Manager* product. He is also known for various presentations and hands-on workshops at the OpenVMS Bootcamp, DECUS, CETS, HP Tech Forum, European Technical Update Days, and various webinars dealing with OpenVMS performance, *Availability Manager*, HP Virtualization strategy around HP SIM/VSE, Blades, and virtual machines. Barry is under exclusive contract to Migration Specialties for emulator and *Availability Manager* services.

The SimH VAX development and test environment was transferred to the Indian development and support group and continues to be utilized.

This article was sponsored by Migration Specialties. Information about Migration Specialties emulator products and OpenVMS services can be found at www.MigrationSpecialties.com