My computer is bigger than yours!

Martin Schwenke

IBM OzLabs Linux Technology Center

<*martins@au.ibm.com*> · <*martin@meltin.net*>

Abstract

Your computer runs Linux. It has 1 CPU, 256MB of RAM, 2 Ethernet interfaces, 2 IDE disks, a CD-ROM drive and a CD-RW drive. You reboot this machine every week or two to try out the latest kernel or to mess with a new revision of the X server. You know you've got `/dev/hda1` and `/dev/hda2`, `eth0` and `eth1`, so the configuration is pretty simple. When something goes wrong with one of the components, it is pretty easy to notice that you have a problem (using, say, `logcheck`) and to remove the cover and point to the faulty component (using your index finger).

My computer runs Linux. It has 32 CPUs, 64GB of RAM, 16 Ethernet interfaces, 540 SCSI disks, a CD-ROM drive and a CD-RW drive. The contract I have with my customer means that I'm allowed about 8 hours of unscheduled down-time per year. Luckily, everything is hot swappable. I need support for persistent device names or things are going to get ugly. I need to anticipate hardware problems, so I need some fancy software that wades through the messages that I get from all of this hardware. When I've detected that something is wrong with one of the components I'm going to need some diagnostic tools that let me figure out the exact physical location of the faulty component.

This paper looks at persistent device naming and hardware inventory requirements for Linux. There are quite a few solutions for persistent device naming but most, if not all, have disadvantages. There are several distinct stages in the hardware inventory process: you need to consider how you're going to obtain, store and use the information about the hardware. Should persistent naming use hardware inventory data or vice-versa? The discussion involves `sysfs`, Open Firmware device-trees, various user-space methods for retrieving hardware inventory data, some serviceability infrastructure that helps to make AIX a serious operating system. . . and much more.

Prelude

Tortoise: OK Achilles, your computer is bigger than mine, so what?

Achilles: It is harder for me to service my machine and so I need to write some tools.

Tortoise: Why didn't you say so?

Achilles: I did!

Tortoise: Oh, sorry! I'm a bit slow. . .

1 Introduction

Linux started off its public life as an operating system kernel for hobbyists, running only on IBM-compatible PCs. In this role, Linux systems generally involved a small amount of hardware chosen from a relatively simple selection. This was a great advantage of Linux - you could run it on old, out-dated systems, with excellent performance. Early Linux kernels focused on providing good interactive performance for the desktop. This was because most people used Linux on low-end, uniprocessor workstations. While an early version of Linux was a useful tool, it was still a toy, compared to some of the commercial Unices.

Since then, Linux has been ported to over a dozen architectures and has support for hundreds of different types of devices. Linux has also grown from running on simple hobbyists' workstations to running on high-end, multi-million-dollar servers. The performance on these high-end servers can now equal that of commercial Unices. Linux is now being taken seriously as an operating system kernel for the enterprise. Large companies are betting their futures on Linux.

However, the move to large servers has been quite rapid, and Linux still doesn't have some of the features that make big companies trust the commercial Unices. Some of the missing features relate to Reliability, Availability and Serviceability (RAS). The rest of this paper discusses some of the RAS features of IBM's AIX operating system, compares them to what is available in Linux and provides details of the current status of the implementation of these or similar features in Linux. The focus is on serviceability and, in particular, hardware inventory management.

The rest of this paper is structured as follows:

- §2 discusses some important RAS features that Linux doesn't fully implement.
- §3 discusses how these features could be and, in some cases, have been implemented.
- §4 briefly discusses how hardware inventory management interacts with persistent device naming.
- §5 provides a brief summary of the current project and considers future directions.

2 RAS features and hardware inventories

This section introduces some interesting facets of RAS, with a focus on serviceability. In this context, Vital Product Data (VPD) is discussed as a vehicle for building a hardware inventory.

2.1 Serviceability

Serviceability is about making it easier to service a machine. One way of doing this is to provide concise, structured information. There are several categories of information that are useful, including information about hardware, software, system configuration and diagnostics. Given the title of this paper, the focus will be almost entirely on information about hardware, with some discussion of system configuration.

2.1.1 Vital Product Data

VPD is information about a piece of hardware that allows it to be easily identified, located, upgraded and replaced. The PCI 2.2 specification [1, p289] says:

> Vital Product Data (VPD) is information that uniquely identifies hardware and, potentially, software elements of a system. The VPD can provide the system with information on various Field Replaceable Units such as part number, serial number, and other detailed information. The objective from a system point of view is to make this information available to the system owner and service personnel.

The PCI 2.2 specification defines a format for VPD to be used in PCI devices. This is a binary format containing ASCII fields. Binary formats are not 'the Unix way', so a textual representation of VPD can be used instead. The format used in this paper is the one used by the AIX `lsvpd` command.

```
*DS 16 Bit SCSI Disk Drive
*AX /dev/scsi/host0/bus0/target8/lun0
*MF IBM
*TM ST318305LC
*YL U1.9-P1/Z1-A8
*FN 09P4435
*RL 43353039
*SN 00043CD2
*EC H11936
*PN 09P4434
*Z0 000003129F00013E
*Z1 0211C509
*Z2 1000
*Z3 02041
*Z4 0001
*Z5 22
*Z6 162870 C
```

Fig. 1: An example of VPD

An example of VPD is shown in Fig. 1. When servicing large machines, we tend to be most interested in the following fields:

DS: Description. Usually displayed as the first item.

AX: Logical name (or AIX name). This allows system diagnostics (for example, kernel log messages) to be associated with physical components. In this particular case, the VPD is generated on Linux and the name is a Linux 2.4 `devfs` device node.

YL: Physical Location. All fields beginning with 'Y' are system specific fields and, in this case, this field is particular to IBM pSeries systems. It can be read (backwards) as SCSI target 8, on integrated SCSI bus 1, on planar (or backplane) 1, in drawer 9, in rack 1.

FN: FRU (Field Replaceable Unit) number. This is a generalisation of a model number, representing a family of models that are interchangeable. For example, an old model may no longer be available, but there may be a newer model with the same FRU number that can be used as a replacement.

RL: Firmware level. Problems may relate to faulty firmware, so a firmware upgrade may be the most useful fix. In this particular case, it has been hex-encoded, so is `C509`.

SN: Serial Number. When we reach into the machine to remove a component, it might be worth checking that we're removing the correct one!

Other fields include:

MF: Manufacturer.

TM: Type/model.

EC: Engineering change level of the board.

PN: Part number.

Z0: All fields starting with 'Z' are device specific. In this case, `Z0` is first 8 bytes of the standard SCSI Inquiry result, hex-encoded. This provides lots of useful information about the device, like the type of SCSI device (disk, CD-ROM, tape, enclosure, ...), number of address and data bits, ...

2.1.2 VPD as hardware inventory

If a system has VPD available for all (important) components, the collection of this information can be referred to as a *hardware inventory*. Two important things to note are:

- A complete hardware inventory must be available before a service event takes place. There is no point trying to retrieve VPD from a device when that device has already failed and is no longer responding.
- It is important that the hardware inventory is persistent across system reboots. Among other things, this allows boot-time hardware failures to be recognised and changes to be managed.

Therefore, when using VPD, hardware inventory management can be separated into several distinct stages:

1. Collection.
2. Storage and change management.
3. Rendering.

2.2 Availability

Service contracts for large systems often stipulate incredibly low down-times. If components can be replaced without shutting a system down, this helps down-time stay low. Therefore, the availability feature of most interest to hardware inventory management is hot-plug.

The event of hot-plug means that a hardware inventory isn't static between system startup and shutdown. A system may run for years between reboots and many hardware components may be interchanged during this time. This means that a hardware inventory needs to be dynamically updated when hardware is changed.

2.3 Reliability

Linux has always been reliable, in the sense that it crashes less than many of its competitors. The aspect of reliability that is of interest with respect to hardware inventory is *persistent device naming*. If hardware changes between reboots it can be very useful for the unchanged devices to be referenced by unchanged logical names.

Interlude

Achilles: So Tortoise, if Linux doesn't have some of this stuff, how can it be used in the enterprise?

Tortoise: Can't talk... hacking...

3 Implementing hardware inventory support in Linux

The differences between Linux architectures make it difficult to use a consistent approach for gathering hardware inventory. A useful approach is to build a set of tools that use a variety of methods to retrieve VPD. The following subsections describe current and future methods. The focus is primarily on VPD collection and touches on storage and change management. VPD rendering is not an incredibly interesting topic: VPD fragments can be concatenated to produce a complete report of hardware inventory, can be re-parsed and rendered via a web-based system, or can be collected via higher-level serviceability tools.

3.1 Open Firmware device-trees, `ibm,vpd` and Linux on pSeries

Most PowerPC systems use Open Firmware (OF) as a pre-boot environment. OF has a similar role to a PC BIOS, only more so — extra features include an interactive command-line interface and a device-tree. The device-tree is a hierarchical representation of the hardware available in the system, and also includes things like boot parameters.

The device-tree on pSeries is augmented with `ibm,vpd` properties, containing VPD extracted from PCI 2.0 and 2.1 devices. At boot-time, the Linux kernel copies the device-tree so it appears as a directory tree under `/proc/device-tree`. Combined, these two facts suggest a very good basis for doing hardware inventory on pSeries. In fact, because the project is currently pSeries-based, the hardware inventory 'database' is currently a copy of the device-tree, in `/var/lib/device-tree`.

At boot time the device-tree is copied and all `ibm,vpd` properties are rendered as text into `linux,vpd` files that sit alongside the originals. Then components other than those with VPD in the device-tree are queried for information and their VPD is placed in appropriate subdirectories of the device-tree in the form of `linux,vpd` files.

While the approach of using a device-tree is useful and easy on pSeries, it is not strictly necessary, so is subject to review if and when inventory management is more fully implemented on other architectures. In particular, the 'database' does not need to have a tree-like structure — mirroring an existing structure is merely convenient, since this obviates the need for links into that structure.

3.2 SCSI

SCSI devices are not PCI devices and don't contain PCI-style VPD, so the VPD needs to be manufactured. Raw VPD for SCSI devices can be extracted using SCSI INQUIRY commands, via the Linux SG (SCSI Generic) interface. This VPD is organised in a device-specific manner, so it is parsed using a template driven system and formatted as textual VPD. There is enough commonality so that all devices encountered so far are covered by a 5 line template file — even though non-IBM disks have not yet been looked at, it is hoped that the template file will stay quite small.

3.3 VPD from PCI devices

VPD is stored in PCI devices in two completely different ways, depending on whether it is done according to revision 2.0/2.1 or revision 2.2 of the PCI specification.

3.3.1 PCI 2.0/2.1

Earlier revisions of PCI had VPD stored in an expansion ROM. The procedure to try and extract VPD looks something like this:

1. Check PCI configuration space to determine whether there is an expansion ROM.
2. If there is a ROM, but it has no address assigned, then assign one.
3. If the ROM is disabled, enable it.
4. Check the PCI data structure in the ROM to see if there is an address for VPD.
5. Read the VPD.
6. Disable, unassigned ROM as necessary.

Currently this works extremely ordinarily. Assigning an address, when one is not already assigned, is not currently attempted, since it is unclear how this should be done. Even ignoring PCI devices with unassigned ROMs, the current implementation is still quite good at locking up machines.[1]

3.3.2 PCI 2.2

PCI 2.2 allows devices to expose VPD via the capabilities list in configuration space. Extracting this VPD is relatively straightforward and is currently done by reading and writing the appropriate configuration space pseudo-file under `/proc/bus/pci`.

It would be interesting to know whether the change in VPD location in PCI 2.2 was due to the difficulty of accessing expansion ROMs on a live system.

3.4 sysfs

3.4.1 Exposing VPD through `sysfs`

As `sysfs`, the filesystem for interacting with devices in Linux 2.5, matures it is possible that VPD will be exposed by it. At one extreme, it is possible that VPD, in a consistent format, such as the binary PCI format or the textual format, could be exposed via `sysfs`. However, the logic to convert device-specific data, such as SCSI INQUIRY data, would amount to kernel bloat — the conversion is not entirely straightforward and can be done in user-space. At the other extreme, the current variety of VPD gathering methods could continue to exist with no support in `sysfs`.

The best approach is probably a compromise between these two alternatives: expose the simple things, such as SCSI INQUIRY output, in `sysfs`. This approach minimises kernel bloat and allows various ad hoc APIs to be replaced by simple file-based interfaces.

[1] Please contact the author if you can explain how to reliably access PCI expansion ROMs from a running Linux system!

3.4.2 `sysfs` as a database format

Since `sysfs` is becoming the standard structure for interacting with hardware on Linux, it may be worth using a `sysfs`-like structure for the hardware inventory database. One useful advantage of the current OF device-tree is that it includes information about the physical locations of major hardware components, which is useful in VPD. How this can be done in `sysfs` needs to be investigated.

3.5 Hardware inventory persistence and change management

A hardware inventory database should persist across reboots. This assists in the detection of changes to hardware at boot time and the management of those changes. The simplest changes involve the addition, removal and replacement of a single component — replacement is a special case involving a single addition and a single removal. However, multiple changes can be more difficult to manage.

Another difficulty is the way Linux devices are named in a way that relates to their order of detection. For example, if a SCSI device is removed (between boots) then, under Linux 2.4 without `devfs`, all subsequent devices will be renamed. Even with `devfs`, if a SCSI host adaptor is removed, Linux will rename all of the SCSI host adaptors that follow, causing all of the devices attached to those adaptors to be renamed. The solution is to do device naming in user-space, using a hardware inventory database, after any changes have been resolved. This is discussed in more detail in §4.

The current Linux hardware inventory implementation simply destroys the any old database at boot time. AIX provides a `diag` command which, when used with the `-a` option, displays changes and provides management options.

Change management options for Linux need to be considered. Existing systems such as Red Hat's `kudzu`, and similar tools, need to be examined, especially to see how well they scale on large systems.

3.6 Hot-plug

Changes is hardware can be tracked by hooking hardware inventory management utilities into Linux's hot-plug infrastructure. Managing these changes is a special case of the change management discussed in §3.5, and should be a little more straightforward to implement.

Note that, on pSeries systems where a device-tree is used as a database, it would be possible to update `/proc/device-tree` on hot-plug events but, since the kernel doesn't need direct knowledge of changes to the hardware inventory database, this would be another case of bloating the kernel for little or no benefit.

4 Persistent device naming

Traditionally, Linux does not have support for persistent naming of devices. Devices are named in the order that they are probed. In Linux 2.4, `defvsd` opens up some possibilities for persistent naming via configurable device aliases. However, the cleanest way of approach persistent naming is via hot-plug, which is most interesting in Linux 2.5. This allows the canonical device name to be persistent, rather than using aliases.

If all devices are added to the system via the hot-plug subsystem, then this is where names can be generated. One project that is attempting to add persistent naming is the 'Device Naming

Project'[2]. This project's `scsiname` utility is implemented entirely in userspace and does its own SCSI INQUIRYs to retrieve information relevant to the naming process. It could, instead, be a thin wrapper around the hardware inventory management system. A similar approach could be used for non-SCSI hardware.

It is possible to implement hardware inventory management on top of persistent device naming, but since the peristent naming needs to know about information that is contained in the hardware inventory, this seems the wrong way around.

5 Conclusions

The first implementation of hardware inventory management for Linux that was done as part of this project was a Perl script that ran on pSeries Linux systems, found all of the `ibm,vpd` properties under `/proc/device-tree`, parsed them and displayed them as text. A subsequent Perl version used a copy of the device-tree and added `linux,vpd` properties for SCSI devices using quite a verbose template format, and was regarded as partially working prototype. The current implementation is about 2000 lines of C and shell scripts and is described in §3. This implementation is quite incomplete and there is a lot of work to be done. We plan on releasing the source for this work under an Open Source license a little later this year.

When the project becomes truly cross-architecture, alternative representations for the hardware inventory database will need to be considered. As mentioned in §3.4, a `sysfs`-based database may be be appropriate. Another possibility is a database that uses relevant parts of the CIM Core Schema[3], which includes objects for PhysicalElements and LogicalDevices.

When complete, an inventory management system, integrated with persistent device naming, will provide major serviceability benefits for large systems running Linux.

Epilogue

Achilles: So Tortoise, how's your Linux hardware inventory implementation going?

Tortoise: Slowly...

[2] `http://www-124.ibm.com/devreg/`
[3] `http://www.dmtf.org/standards/documents/CIM/CIM_Schema27/CIM_Core27.pdf`

Dedication

For Uncle Stan: Stanley Roy Fisher, 1920-2002. A hacker, before his time...

Thanks...

- Greg Rodgers, Paul Mackerras, Keith Matthews and Angelo Salza for getting me started on various bits of this.
- The rest of the OzLabs team.
- Various IBM LTCers.
- Several AIX developers who have provided important pieces of information.
- Mel.

Trademark acknowledgements

References

[1] PCI Local Bus Specification. Release 2.2. PCI Special Interest Group. December 18, 1998.