Linux hardware inventory: Current reality, future possibilities

Martin Schwenke

IBM OzLabs Linux Technology Center

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

Abstract

The `lsvpd` and `lscfg` commands are used on AIX® systems to list interesting information about the hardware that makes up a system. These commands have been partially reverse-engineered and reimplemented for Linux® running on PowerPC®. The Linux versions have reached a point where they are likely to provide useful information for an IBM service engineer who is familiar with the AIX commands, but comes across a pSeries® machine that is running Linux.

The first part of this paper covers progress so far: the early days of the project - this should take a few hours; a Perl prototype - Perl probably isn't available early enough it the boot sequence, so we'll call it a prototype; more experiments in C and Bourne shell - success at last; and packaging issues - would you like `find` with that?

The second part of this paper covers possible futures: including cross-architecture hardware inventory and device-naming. Some of this involves changing the direction of various parts of the project and looking at areas that are outside the scope of the project.

1 Introduction

Linux distributions generally provide simple mechanisms for detecting what hardware components are attached to a machine. For example, Red Hat uses kudzu[1] and SuSE uses hwinfo. These systems tend to provide a database that persists across reboots, along with simple forms of change management. However, they don't provide platform specific information that allows a hardware engineer to easily locate particular components, or any detailed information about components, such as Vital Product Data (VPD). The main focus of these tools is convenience — ensuring that hardware is properly detected and is configured as automatically as possible.

AIX has a large, mature hardware inventory system. The Object Data Manager (ODM) is a store containing persistent information about hardware components, including generic, static information that is installed along with device

[1] `http://rhlinux.redhat.com/kudzu/`

drivers, as well as dynamic information about the current configuration of components. While certain changes in hardware configuration are handled using sensible defaults, more complex changes can be managed using the `chdev` and `diag` commands. The `lsvpd` and `lscfg` commands allow information about components to be retrieved. `lsvpd` lists VPD in a compact format that is suitable for both human browsing and parsing by higher-level system management utilities. `lscfg` lists hardware configuration information in a human readable format that can include VPD and platform-specific information. The main goal of these tools is to improve serviceability, one of the components of RAS.[2]

This paper describes the history, progress and changing aims of the Linux lsvpd project. The resulting software currently only works on modern IBM pSeries (PowerPC-based) systems. The first section covers the period between the project's inception and the present. The second section speculates on the project's future.

[1] is an earlier paper describing the project from a different perspective, including more background on its RAS goals.

2 Are we there yet?

The Linux lsvpd project has been through a number of distinct stages that are summarised in this section.

2.1 This should take a few hours!

This project started in about May 2001 when Greg Rodgers, a long-term IBMer associated with pSeries, but now working on Linux, pointed out that Linux on pSeries needed an `lsvpd` command. He pointed out that the Open Firmware (OF) device-tree, imported by the Linux PPC kernel into `/proc/device-tree`, contained 'properties' called `ibm,vpd`. These files, containing binary data, simply needed to be converted to a textual format similar to that produced by the AIX `lsvpd` command, as shown in Figure 1.

```
*DS 2 RIO-PCI COPPER
*SN YL1182206012
*PN 53P3820
*CC 2887
*FN  53P3800
*VK RS6K
*YL U0.4-P1.1
```

Fig. 1: An example of VPD

[2] Reliability, Availability and Serviceability.

The important fields shown in Figure 1 are described as follows:

DS: Description. Usually displayed as the first item.

SN: Serial Number. It is useful to confirm the serial number of a component before removing it, when possible.

PN: Part Number. Knowing the history of a part may help to explain certain types of faults.

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.

YL: Physical Location. All fields beginning with 'Y' are system specific fields and, in this case, this field is particular to IBM pSeries systems. This example can be read (from right-to-left) as extender 1, on planar (or backplane) 1, in drawer 4, in rack 0.

This task began as a simple reverse engineering, parsing and pretty printing job, and proceeded fairly quickly. A single Perl script seemed the perfect tool for the job and there wasn't even a need to understand the VPD contained in the files. To make things even easier, it was soon 'discovered' that the data in the `ibm,vpd` files wasn't in a weird proprietary format, but was in a format detailed in the PCI specification [2]. The job was all-but complete.

As usual, with any reverse engineering job, things weren't quite that simple. Greg compared our script's output with that from a machine running AIX and noticed that 'things like SCSI disks' were missing. Some e-mail to and from members of the AIX lsvpd team confirmed that it was to be a much bigger job: the OF device-tree doesn't contain VPD for everything, especially things like SCSI devices - these things need to be 'probed' separately and the VPD needs to synthesised from available information.

2.2 Snazzy SCSI solutions

SCSI devices seemed to be the most useful class of components that were missing VPD. Therefore, we decided to augment the device-tree with VPD files for SCSI devices, primarily as a 'proof of concept'. An important decision was made to avoid trying to update `/proc/device-tree` - that would involve kernel modifications, and this was something the kernel didn't need to know about. Therefore, the device-tree was copied to `/var/lib/device-tree`, so any extra VPD could be added there. Change management would be considered later.

Some experimentation with SCSI Inquiries, using the `sg_inq` from the sg3_utils[3] package, and comparing the output with that of the AIX `lsvpd` command, suggested the use of templates. The first 32 bytes of the 'Standard Inquiry' output are standard, with higher bytes depending on the model of a device. Extended VPD (EVPD) pages are also available, depending on the model. Luckily, much of the non-standard information is almost identical across all disks certified for

[3] `http://www.torque.net/sg/`

use in pSeries machines, so less than a dozen templates were needed, with most of these sharing common fields.

Figure 2 shows the output from a SCSI Standard Inquiry and Figure 3 shows the corresponding VPD.

```
00000000  00 00 03 02 9f 00 01 3a  49 42 4d 20 20 20 20 20  |.......:IBM     |
00000010  49 43 33 35 4c 30 33 36  55 43 44 32 31 30 2d 30  |IC35L036UCD210-0|
00000020  53 35 42 53 56 4d 46 39  39 33 31 38 30 37 4e 34  |S5BSVMF9931807N4|
00000030  39 30 38 20 20 20 20 20  0c 00 00 00 00 00 00 00  |908     ........|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000060  20 20 30 30 37 35 30 32  31 37 32 00 30 30 30 31  |  007502172.0001|
00000070  32 32 30 39 50 33 39 32  33 20 20 20 20 20 48 33  |2209P3923     H3|
00000080  32 30 35 31 20 20 20 20  30 37 4e 37 30 37 30 20  |2051    07N7070 |
00000090  20 20 20 20 46 38 30 34  37 30 20 20 20 20 32 30  |    F80470    20|
000000a0  30 32 00 00                                       |02..|
000000a4
```

Fig. 2: Hexadecimal dump of SCSI Standard Inquiry output

```
*DS 16 Bit LVD SCSI Disk Drive
*AX /dev/sda
*MF IBM
*TM IC35L036UCD210-0
*YL U0.4-P1-I1/Z2-A8
*FN 09P3923
*RL 53354253
*SN VMF99318
*EC H32051
*PN 07N7070
*Z0 000003029F00013A
*Z1 07N4908
*Z2 0075
*Z3 02172
*Z4 0001
*Z5 22
*Z6 F80470
*Z7 500507620CC4AC8E
```

Fig. 3: VPD for a SCSI disk

Interesting fields in this VPD, include:

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, so the device-name is for the first (logical) SCSI disk.

YL: Physical Location: SCSI target 8, on integrated SCSI bus 2, on I/O adapter 1, on planar (or backplane) 1, in drawer 4, in rack 0.

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 `S5BS`.

MF: Manufacturer.

TM: Type/model.

EC: Engineering change level of the board.

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, ...

Z7: Worldwide ID (WWID) of the disk. AIX `lsvpd` doesn't show this information, but it is useful for persistent device naming, so has been included in the Linux version.

2.3 Perls of wisdom...

The single Perl script was split into two scripts: `update-device-tree`, used to copy the `/proc/device-tree` and add `linux,vpd` files for the SCSI devices, and `lsvpd`, which printed a header and rendered the `ibm,vpd` and `linux,vpd` files. When complete, the core parts of the package contained about 2000 lines of Perl code. Most of the actual code was in Perl library modules. Interesting aspects of the implementation include:

- The Perl `unpack` function proved very useful for parsing binary data, including PCI VPD chunks.
- SCSI Inquiries were initially done by running the external `sg_inq` command and parsing the hexadecimal output.
- SCSI Inquiries were reimplemented using Perl's `ioctl` function. The `pack` function was used to construct arguments to `ioctl` and output was parsed using `unpack`. Formats for `unpack` were constructed from fairly verbose templates, represented using Perl hashes that were stored in configuration files.
- Determining the physical location (`YL` field) for a SCSI device involved matching a logical device, such as `/dev/sda`, with the corresponding node in the OF device-tree for the 'physical' SCSI host adapter. Theoretically, the best way of doing this involves matching PCI bus, device and function data for the adapter. The implementation involved (at least) two problems:
  - While it is easy to determine the SCSI host adapter associated with a device, it is more difficult to get the PCI data for the adapter. The only universal method under Linux 2.4 is to parse the host adapter information files, such as `/proc/scsi/sym53c8xx/0`. However, the format of these files is driver specific, so more templates were needed.
  - Many of the 64-bit pSeries machines can have more than 256 PCI buses, even though the Linux kernel usually stores the bus number as an `unsigned char` (an 8-bit quantity). These machines are referred to as 'bignum machines'. The Linux 2.4 PPC64 kernel is hacked so that PCI bus numbers are stored variously as either `unsigned char`,

`int` or `unsigned int`. This inconsistency, and a lack of any information in the OF device-tree about buses with numbers greater than 8-bits, meant that the matching was not possible. Two kernel patches were required:

* A `linux,global-number` property (later called `linux,phbnum`) was added to the OF device-tree node for each PCI host bus. The kernel uses these numbers as the top 24 bits of PCI bus numbers.

* The sym53c8xx SCSI driver, used almost exclusively in the relevant machines, was patched so the host adapter information files would always contain a 32-bit quantity (via an `unsigned int`).

In about June 2002, it was realised that the lsvpd package might be required in single-user mode, to aid fault diagnosis, and very early in the boot sequence, so various pieces of information could be used as inputs to persistent device naming. Since Perl resides on the `/usr` filesystem, and only a root filesystem might be available, Perl was considered unsuitable for ongoing development. Therefore, development on the Perl version ceased and the Perl version was retrospectively considered a prototype.

Several lessons were learned from the Perl prototype, including:

- A scripting language was appropriate for a large part of the project, since there was a lot of fairly simple string handling and file input and output.

- The templates used to parse SCSI Inquiry data were too verbose and difficult to manage.

- The approach of using an augmented copy of the device-tree as a database seemed to work, as did the split in functionality between `lsvpd` and `update-device-tree`.

2.4 C and shell (by the seashore?)

A decision was made to continue using a scripting language as much as possible, and to write helper commands in a compiled language for specific purposes that couldn't be satisfied using the scripting language and the available external commands. The only scripting language that is generally available in Linux, especially without a `/usr` filesystem, is a Bourne-compatible shell. Since `bash` is used on most Linux systems, and provides built-in support for things like arithmetic, it was chosen as the scripting language.

The next choice to be made was that of the compiled language. While C++ provides a nice selection of classes that might make life easier, it also tends to produce rather large executables, which may not be appropriate for an early-boot environment. Therefore, C was chosen as the language for writing helper commands. In an effort to avoid the 'not invented here syndrome' to some extent, the glib-2.0 utility library was selected for use.

2.4.1 Updating `update-device-tree`

One change in approach centred on the fact that `update-device-tree` would be run relatively rarely, while `lsvpd` would be run quite often, with a real person waiting for output. This suggested that `lsvpd` should be as fast and simple as possible - it should simply find `linux,vpd` files and print their contents. The rendering of `ibm,vpd` files would be done by `update-device-tree`.

One important issue would be guaranteeing availability of all commands used on the root filesystem. Although `sed` appears in `/bin`, and would be indispensable for simple parsing, commands such as `find` and `sort`, usually found in `/usr/bin` were also required. Therefore, a dependency on busybox[4] was added, since it could provide the subset of functionality of these commands that was being used.

2.4.2 Rendering `ibm,vpd`

The first task attempted was to write a C program to render PCI VPD. More particularly, the `ibm,vpd` files can contain several PCI VPD chunks, each preceded by a four byte length (in network order). This resulted in two modules:

`pci_vpd_to_text.[ch]`: Takes a standard PCI VPD chunk and renders it as a text string.

`ibm_vpd_render.c`: Reads an `ibm,vpd` file, extracts the individual VPD chunks, hands each of them to `pci_vpd_to_text()` for parsing, and combines the results, inserting a delimiter before the output for each chunk.

Later, most of the functionality of the latter module was separated into a new module called `ibm_vpd_to_txt.[ch]` and temporarily used by another module until 'a better way' was found.

2.4.3 Rendering SCSI VPD

The next task was to implement production of SCSI VPD. This involved reintroducing the use of `sg_inq`, since it was already the most appropriate 'helper utility' for retrieving SCSI Inquiry data. A small patch to `sg_inq` that added a `-r` option to produce raw, binary output, instead of formatted hexadecimal text, was accepted into sg3_utils 1.01.

Several stages were used to produce the SCSI VPD:

`scsi_vpd_scan`: A script that lists all available SG devices and iterates over them using `scsi_vpd_inquire`.

`scsi_vpd_inquire`: A script that takes a generic SCSI device and an output directory, and dumps all available Inquiry data for the device into that directory.

`scsi_vpd_std`: A C helper that extracts useful information found in the first 32 bytes of the Standard Inquiry, and produces a file for each available field.

[4] `http://busybox.lineo.com/`

`scsi_vpd_custom`: A C helper that uses templates to extract other fields from the rest of the Standard Inquiry and other available Inquiry data. An example template is shown in Figure 4. The '`*`' matches any model number. The Standard Inquiry is parsed, from byte 32 onwards, yielding the `RL` field (4 bytes), `SN` (8 bytes), `Z1` (12 bytes), skipped data (42 bytes), `Z2` (4 bytes), and so on. Page `0x83` of EVPD has the first 8 bytes skipped, before the next 8 bytes are extracted into the `Z7` field. The templates are very compact and easy to parse, as well as being effective. Due to the wild-carding facility, only five templates are currently needed.

Field generation: The `YL` and `AX` fields are generated by `update-device-tree`.

`scsi_vpd_render`: A script that collects together the individual fields produced by the previous three stages, and prints them in a consistent order.

```
IBM;disk;*;
inquiry=RL:4,SN:8,Z1:12,_:42,Z2:4,Z3:5,_:1,Z4:4,Z5:2,FN:12,EC:10,PN:12,Z6:10,_:4;
0x83=_:8,Z7:8
```

Fig. 4: Template for most IBM SCSI disks[5]

2.5 Enter `lscfg`

In August 2002, Todd Inglett, an IBMer working on related utilities, contributed a version of `lscfg` that simply ran `lsvpd` and pretty printed the output. Figure 5 shows some output for a SCSI disk, from the current Linux version of `lscfg`.

```
  sda              U0.4-P1-I1/Z2-A8  16 Bit LVD SCSI Disk Drive (36400 MB)
        Manufacturer.............IBM
        Machine Type and Model......IC35L036UCD210-0
        Device Specific.(YL)........U0.4-P1-I1/Z2-A8
        FRU Number..................09P3923
        ROS Level and ID............53354253
        Serial Number...............VMF99318
        EC Level....................H32051
        Part Number.................07N7070
        Device Specific.(Z0)........000003029F00013A
        Device Specific.(Z1)........07N4908
        Device Specific.(Z2)........0075
        Device Specific.(Z3)........02172
        Device Specific.(Z4)........0001
        Device Specific.(Z5)........22
        Device Specific.(Z6)........F80470
        Device Specific.(Z7)........500507620CC4AC8E
```

Fig. 5: `lscfg` output for SCSI disk

[5] Templates are actually single-line, but this one has been split for presentation purposes.

2.6 Cross platforms?

At this stage it looked like it might be time for the project to move from just pSeries, to working on any Linux architecture. This would mean breaking some ties with the OF device-tree, and implementing helper programs to retrieve VPD directly from PCI adapters, rather than relying on the device-tree to conveniently dish them up. Two such helpers were (partially) implemented:

`pci_vpd_cap_grab`: This utility successfully extracts PCI 2.2 style VPD via the capabilities list in an adapter's PCI configuration space, producing `ibm,vpd`-style output. This form of output was chosen, rather than a single VPD chunk, because the PCI 2.2 specification is a little unclear about whether an adapter can contain multiple VPD capability items.

`pci_vpd_rom_grab`: This utility attempts to extract PCI 2.0/2.1 style VPD from a PCI expansion ROM, but usually succeeds in locking up a machine! It seems certain that kernel support will be needed to access the ROMs - the most useful idea would be to have sysfs[6] optionally contain binary blobs containing data from any PCI expansion ROMs. This has not yet been implemented.

Without the above basic functionality, making `lsvpd` work across architectures is not a reachable goal.

2.7 Testing times

In February 2003, IBM hardware test laboratories in Austin wanted to test new pSeries hardware configurations, running Linux. They also wanted a usable version of `lscfg` to assist with the testing, and they wanted that version to be available as part of the Linux distributions most likely to run on the hardware. They wanted to complete most of their testing by the end of March. This would be the first time the lsvpd package would be 'used in anger'. At this time `lscfg` was a pretty printer and was missing most of the functionality of its AIX role model.

During and since this time, most of `update-device-tree` and `lscfg` have been incrementally rewritten. Many additions to `lscfg` involved directly rendering parts of the device-tree - rather than re-rendering `lsvpd` output, `lscfg` became more closely tied to the device-tree. So much for going cross-architecture!

Early on, it was noticed that the patch for the sym53c8xx SCSI driver hadn't been merged, and it wasn't likely that this would happen very quickly. The template scheme was augmented with information about IRQs, and they were incorporated to provide more reliable matching of SCSI host adapters. It was ugly, but good enough. There was still no consistent and reliable way of finding the PCI information for Ethernet adapters, so interface numbers were simply allocated sequentially starting at 500, to avoid any confusion caused by their names accidently matching those assigned by the operating system.

[6] sysfs is a Linux-specific device-tree implemented in Linux $\geq$ 2.5, allowing drivers to export information in a uniform way.

The first usable version (0.8.4) was publicly released as part of the SourceForge.net linux-diag project[7] in May 2003. Since then progress has continued.

`scsi_vpd_scan` has been obsoleted. Its functionality has been replaced by two new functions internal to `update-device-tree`: `scsi_list_devices` and `scsi_do_device`. The idea is that devices might be listed at boot time, but in Linux $\geq$ 2.5, with usable hotplug, devices will often need to be processed separately, so functions like `scsi_do_device` can be separated into a library and used by multiple scripts. A quick browse of `scsi_vpd_inquire` and `scsi_vpd_render` show that they can be simplified and replaced by similar, smaller library functions, instead of being stand-alone scripts.

Support for using sysfs to determine PCI information for adapters has been added. This works for all adapters, not just SCSI, and doesn't require templates, since the required information is explicit in the structure of sysfs. When available, sysfs is also used to efficiently list various types of adapters.

Support for PCI domains[8] has also been added in Linux 2.5, so issues relating to bignum machines have been solved there. `update-device-tree` has been updated to support PCI domains for matching PCI adapters. The old script for determining SCSI adapter PCI information has had PCI domain support retro-fitted, even though it is now used only when sysfs is unavailable.

2.8 Baggage claims

Packaging lsvpd has been an interesting exercise. As mentioned above there are dependencies on sg3_utils and busybox (for `sort` and `find`). Currently SuSE Linux rolls sg3_utils into their scsi package, and all of the programs live on the `/usr` filesystem. Also, SuSE's current SLES release doesn't include busybox. The idea of including extra commands in a future release, and moving others, has been discussed with SuSE representatives.

2.9 Summary

The package currently contains about 1500 lines of bash script and 1500 lines of ANSI C source. Keeping it this small has involved constant tweaking and refactoring, resulting in code that is more maintainable and usually more efficient.

3 Where to now?

This section discusses directions that lsvpd needs to go in... and directions that simply seem worth considering...

[7] `http://sourceforge.net/projects/linux-diag/`

[8] PCI domains are groups of PCI buses, identified by what used to be the top bits of the bus numbers.

3.1 Cross platforms?

Will lsvpd go cross-architecture? The various components need to be considered:

`update-device-tree`: This command has 'device-tree' in its name, so hardly seems like a candidate for architectures with an OF device-tree! However, there is no reason why much of the scanning code can't be reused on architectures where components have to be probed directly (as is already done for SCSI devices) or there is another mechanism for getting information about components (such as ACPI).

Since much of the VPD in PCI adapters seems to be stored in expansion ROMS, a workable replacement for `pci_vpd_rom_grab` needs to be found, probably using sysfs, as mentioned above. However, since expansion ROMs can have non-trivial sizes, copying them into sysfs is probably not an option for smaller embedded systems where memory is at a premium.[9] One hopes that memory size scales roughly linearly with the number of PCI adapters on a system!

`lsvpd`: There is nothing here that particularly requires a device-tree — `lsvpd`'s main job is to simply find all of the `linux,vpd` files in a 'database' and print their contents.

`lscfg`: Many recent changes have made extensive use of the device-tree or have focused on presenting platform specific information. At the moment a `-p` option prints a separate platform-specific section, but the generic section is still tied to the device-tree. Factoring can be done to more cleanly split out the architecture-specific logic.

3.2 Persistence in device naming

Linux currently lacks persistent device-naming. In Linux 2.4, devfs is meant to keep names reasonably persistent. In the case of SCSI devices, names are given according to host, channel, device and LUN numbers. However, the numbers for SCSI host adapters are still allocated in 'probe order' by the kernel, so removing or adding adapters can have a major effect on the name of attached devices.

One part of ensuring devices are named in a persistent way is to use information about the hardware. The hardware inventory kept by the lsvpd package is one possible source of such information. VPD fields, or combinations thereof, could be used as keys in a lookup table of device names. One possibility is to use component identifiers, such as `Z7` (WWID) for SCSI devices or `(MF, TM, SN)` for other devices (and SCSI devices without a WWID). Another option is to use slot identifiers, such as `YL`.

Making the whole hardware inventory database (the device-tree, in this instance) available for device naming doesn't seem necessary. It should be possible to use a subset containing just the relevant information.

[9] Such systems are unlikely to be very complex, or change very much, so the same RAS issues aren't applicable.

3.3 Coping with change

One task of a hardware inventory system is to manage changes that occur in the hardware configuration. The most important changes relate to device naming: when physical devices are changed, how does the operating system's view of them change. Implementing a change management system requires:

- a history mechanism;
- heuristics for handling different types of changes; and
- software utilities for handling cases where the heuristics don't apply or need to be manually overridden.

An example of where the choice of heuristics is very important is when two adapters are swapped between two slots (physical locations) — should the adapters retain their old names, since their individual identities (such as `(MF, TM, SN)`) are unchanged, or should their names be swapped, since it may be preferable to associate names with slots?

AIX currently uses the hardware inventory database to manage such changes, as do Linux tools like kudzu. These need to be investigated.

Currently lsvpd maintains a change history by simply keeping all old augmented device-trees, from previous runs of `update-device-tree`. The `lsvpd` and `lscfg` commands can accept an alternative device-tree directory via a command-line option. This history mechanism is too simplistic, but it is better than nothing.

3.4 Supporting large systems

Linux kernels, up to and including Linux 2.5[10], impose some fairly serious limits on the number of devices that can be supported. For example, only enough major numbers are allocated to cope with 128 SCSI disks. A more universal limitation is imposed by the use of 8-bit device minor numbers, since many device classes are only allocated a single major number. Also, tools such as `sg_scan`, which is used by `scsi_list_devices` to enumerate SCSI devices when devfs is not available, are currently limited to a maximum of 128 devices.

On large systems these limits are unreasonable. One possible solution is for the Linux kernel to adopt larger (32-bit?) device minor numbers but, since such a change would be ubiquitous, it might be difficult to push through. Another possibility is for device numbers to be simply allocated from the current 16-bit major/minor space.

Also, on Linux 2.5, some drivers, such as the sg driver, still need to have sysfs support added. This may provide a better way implementing `scsi_list_devices` on systems with sysfs, but still requires the kernel to provide a mechanism for dealing with more than 128 or 256 devices.

lsvpd is inherently able to cope with large systems, but the required infrastructure is not yet available.

[10] At least those 2.5 kernels released at the time this paper was written.

3.5 Bolting on back-ends

In the long-term, a device-tree may not be the best format for a hardware inventory database. This is especially true if the package is to be used on architectures without an OF device-tree. Also, the current scheme is not incredibly efficient since `lsvpd` and `lscfg` search a reasonably deep directory tree for `linux,vpd` files. Once created, the `linux,vpd` files could be stored in an alternative structure for printing by `lsvpd` and re-rendering by `lscfg`. On IBM pSeries machines, `lscfg` would still require an OF device-tree as a source of supplementary and platform specific information. Similar requirements may exist for other architectures.

Alternatives include:

- A possibly different directory structure for each architecture. This is possible by using different setup and rendering hooks according to the architecture.
- A sysfs-based solution. This could be either an augmented sysfs device-tree, or a completely different structure with links into the sysfs device-tree.
- Using an existing hardware inventory database, such as that used by kudzu.
- Using a Common-Information-Model (CIM) Object Manager (or CIMOM). The CIM Core Schema[11] contains objects such as PhysicalElement and LogicalDevice. It is unclear whether this schema would be rich enough to accommodate all of the information currently handled by the lsvpd package. CIMOMs also seem to have quite a large footprint.

4 Conclusions

The Linux lsvpd project has progressed from being a quick hack to a reasonably mature serviceability package for Linux running on IBM pSeries machines. It includes utilities that are potentially useful outside the current project. Parts of the software also have the potential to be used on other platforms. Support for Linux $\geq$ 2.5 is being added as useful features become available. Relationships between lsvpd and other hardware inventory systems, such as kudzu need to be investigated, as do alternative database back-ends. A hardware inventory database, such as that used by the lsvpd package, should be put forward as an information source for persistent device naming. Change management also needs to be addressed.

[11] `http://www.dmtf.org/standards/documents/CIM/CIM_Schema27/CIM_Core27-Final.pdf`

Thanks...

- The IBM OzLabs team — an amazing group of people to work with — and a host of other IBMers.
- Patrick Mochel for useful discussions about sysfs.
- Mel.

Legal Statement

References

[1] Martin Schwenke. My computer is bigger than yours! In Linux.Conf.AU 2003 <http://linux.org.au/conf/2003/>, January 2003. Paper and slides also available from <http://meltin.net/people/martin/publications/bigger.html>.

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