Linux performance tuning using Powertweak

Dave Jones davej@suse.de

1 Abstract

Powertweak is the first multi-purpose performance tuning program for Linux. Whilst it shares its name with a similar program for Microsoft Windows, its code has been written from scratch, and provides very different levels of functionality.

2 Introduction

This project began early in the 2.2 kernel development. There is a feature present in 2.2.x called "Optimize PCI bridges". It provided functionality that enabled bits in hardware registers in PCI host bridges, allowing for things like enabling PCI bursting, posted writes etc.

Annoyed that this did not support the chipset I had in my workstation, I started hacking support for it into the existing code. The code only allowed for a maximum of 5 pokes per chipset due to a struct size limitation, and also only supported on/off configuration. Some registers may take several bits to define a choice of more than two options.

After a comprimise, and picking the 'best' 5 registers to poke, the patch was released for critique on linux-kernel. The patch never got integrated. The reasoning for this, was that it was suggested that the whole functionality would be better done in userspace.

3 Early development

In April of 1999, work began coding a replacement for this function using PCILIB, the userspace PCI access library. Things got going quite quickly.

By moving the feature to userspace, additional functionality could be added.

1. The number of pokes per chipset was now dynamic. The structs were replaced with parsing of a file describing the register layouts, and arrays were built at runtime depending on what PCIset was detected.
2. Support for multi-bit register settings per tweak
3. Extra control, now individual settings could be switched on, as opposed to the 'switch on everything' approach of the original.

The whole feature became more configurable for end users. There may have reasons for the pokes being in the disabled state, for example, to work around problems with specific hardware.

Towards the end of April, the program was nearing what I had set out for 1.0 functionality. Discussion took place to include the program (Then dubbed 'TunePCI' in pciutils. Then, things changed dramatically.

The developer of a Windows shareware program called "Powertweak" was looking for someone to port his program to Linux. Investigating the program revealed it only really had two functions.

- Tuning bits in CPU registers.
- Tuning bits in PCI registers.

Paranoia took me at this point. The Windows program supported many more PCI chipsets than my program. Worrying that someone may come along and do this port, dwarfing my program into obscurity, I contacted the programmer, and made the offer to do the port. He accepted, and I began work on the port.

s/TunePCI/Powertweak/

And Powertweak 0.1.0 was born.

4 Early versions

I was not prepared for what happened next. Hundreds of mails requesting features (But very few sending patches). Over the next few revisions, support was added for lots of other PCI chipsets, and tuning of sysctls was added.

Work began on a GTK based GUI for adjusting the numerous options in the config file. This was completed quite quickly, but by the fifth revision of the program, things were starting to show signs of strain.

Very little design had been done on the program. What had started as a hack moving the kernel space function to user space, had been extended as hack on top of hack. A concerted effort was spent for another few revisions trying to clean up some of the design mistakes, which was partly successful, but eventually it became obvious that things had gone too far, and a complete rewrite was necessary.

This had happened at the same time that Arjan van de Ven had proposed a patch allowing the GUI to create options for sysctl tuning from an XML config file. Not knowing a lot about XML, and only hearing bad things about it, I was initially hesistant about accepting the patch.

After inclusion in one of the last revisions before the rewrite, it became clear that it would save a lot of development time.

5 Redesign decisions

5.1 Layered model

The first decision decided upon was that in order to prevent the program becoming the monolithic mess the original did, a layered model was drawn up. Specific functionality would be provided by backends

5.2 Extra usage of XML

After the success of the XML parser for the sysctl tuner, the idea of file-based tweaks, as opposed to having definitions in the program was decided to be a good thing. Much of the XML parsing code could be reused, allowing implementing additional backends in a much shorter time than before.

5.3 Client/Server architecture

A choice was made to move the routines involving actual adjusting of values (Such as PCI hardware poking, sysctl changing etc) out of the GUIs, into a daemon started at boot time. The daemon would also provide additional functionality in a later revision.

5.4 Modular backends

The multiple standalone backends of Powertweak are built as shared libraries, loaded at runtime, and unloaded where not applicable.

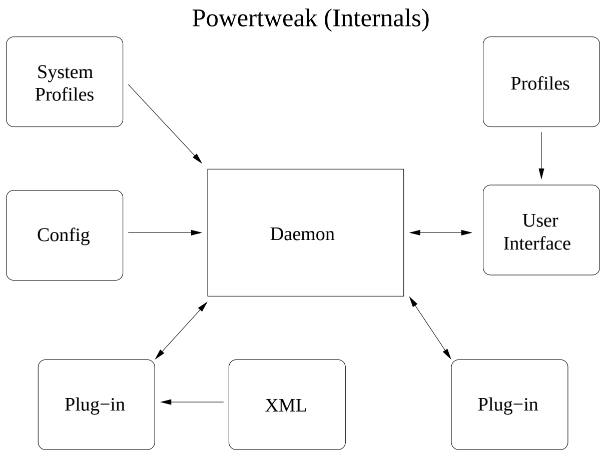

Figure 1: **Powertweak Internals**

6 Current available backends

6.1 sysctl

The various tunable kernel settings through /proc/sys are supported with this backend. Additional entries can be quickly added by adding extra XML entries. XML entries for this backend typically look like..

```
<PROCENTRY>
<MENU>Virtual Memory</MENU>
<SUBMENU>bdflush</SUBMENU>
<TAB>bdflush</TAB>
<NAME>Activate bdflush when % dirty</NAME>
<CONFIGNAME>vm/bdflush:1</CONFIGNAME>
<TYPE>Slider</TYPE>
<LOW>20</LOW>
<HIGH>90</HIGH>
<ELEMENT>0</ELEMENT>
<FILE>/proc/sys/vm/bdflush</FILE>
</PROCENTRY>
```

6.2 PCI

The feature that started the whole project, the PCI backend is still based upon PCILIB, and now parses XML configuration for chipsets. On startup, the PCI bus is scanned for devices, and for each one found, an XML file with a corresponding ID will be searched for. If a match is made, the file is parsed. The PCI XML files contain multiple REGISTER records.

```
<REGISTER base="0x50" bit="5">
<FRAME>Cache controller tweaks</FRAME>
<TYPE>Checkbox</TYPE>
<WIDGETTEXT>Linear burst</WIDGETTEXT>
<DESCRIPTION>Enable linear
bursting</DESCRIPTION>
<CONFIGNAME>LINEARBURST</CONFIGNAME>
</REGISTER>
```

Additionally, multi-bit features may be set using bitmask entries.

```
<BITMASK>11000000</BITMASK>
<ONBITS> 10000000</ONBITS>
<OFFBITS>00000000</OFFBITS>
```

Finally, the PCI backend also allows modification of device latencies. Some of the features provided by the PCI backend are dangerous. Random experimentation will likely lead to a crash, or possibly data corruption. This feature is provided for power users, and people researching PCI related issues.

6.3 Block layer elevator

The elevator algorithm used in the block layer reorders request queues to minimise disk head seeks. This backend allows the length of the reorder queue to be adjusted to improve latency.

6.4 x86 model specific registers.

Pentium class x86 processors typically have model specific registers (MSRs) which contain additional bits which may be adjusted to provide performance features. These may not have been enabled at boot time by the BIOS if for example, the CPU is newer than the BIOS. It's also not uncommon for BIOS vendors to simply not implement enabling of some features. Until the 2.4.0 kernel, this feature was not possible to implement. MSRs must be accessed using a kernel-space driver.
(*nb*, also backported to 2.2.18)

This is another backend that relies upon XML to define the register layouts. Each XML file describes model specific register layout, containing any number of MSR records, which are defined thus:

```
<MSR register="0xC0000080" bit="1">
<NAME>Data prefetch enable</NAME>
<CONFIGNAME>MSR_AMD_K6_3_DPE</CONFIGNAME>
<TYPE>Checkbox</TYPE>
<DESCRIPTION>Enable data prefetching.
Cache misses initiated by a memory
read within a 32 byte cacheline are
conditionally followed by cacheline
fetches of the other line in the 64 byte
sector</DESCRIPTION>
</MSR>
```

As with the PCI backend, multi-bit features may be set by using bitmask entries.

7 Other features

7.1 Profiles

The profiles feature of Powertweak allows a collection of settings to be provided for setting a policy on how the computer on which it runs will be used. For example, a webserver profile could be loaded, which contains optimal settings for all backends.

8 Current developments

8.1 Dynamic bus speed adjustment

This functionality has been implemented in several CPUs from several different vendors (With all implementations completely incompatable with each other)

PowerNOW!
- AMD K6-2+, K6-3+, Mobile Athlon.
Longrun.
- Transmeta Crusoe
Longhaul.
- VIA Cyrix III / C3
Speedstep.
- Intel Mobile CPUs.

nb In addition to the x86 implementations, CPU clock speed scaling is also available on other architectures such as ARM, and PowerPC.

Programming information has been made available for all of these implementations except for Intel Speedstep. For this reason, it is currently unsupported.

By adjusting CPU MSRs, the bus speed can be dropped dramatically. Powertweak uses a kernel module to create a /proc/sys/cpu/0/frequency sysctl entry. This can be altered at any time by the user with a command such as

```
echo 300 > /proc/sys/cpu/0/frequency
```

Alternatively, it may be adjusted periodically by the daemon to values dependant upon user selected monitoring criteria. For example:

- If load is significantly low for a prolonged period of time
- When battery power reaches a critical level in a laptop.
- When switching from battery to mains power (or vice versa)
- When temperature reaches user defined levels.

The policy can be changed by switching strategy modules in the daemon configuration.

Some of these implementations also allow the core voltage supplied to the CPU to be altered. This is a complicated area, due to several different VRM (voltage regulator module) specs in use. To adjust voltage safely, the version of VRM in use must be detected, and adjusted for accordingly. By adjusting the voltage and the frequency, even lower power consumption rates are possible than by dropping frequency alone.

Some implementations of clock scaling have additional caveats. For example the AMD PowerNOW! implementation requires that the PCI bus decoding be temporarily disabled during the speed change. Adjusting the speedon ARM SA11x0 also requires changing the clocks of peripherals clocked off the CPU clock, such as the SDRAM controller. For these reasons, the adjusting code moved to a kernel module.

9 Acknowledgements

The Powertweak Linux project would have been difficult to make possible without the contributions of Arjan van de Ven, Philipp Rumpf, Janne Penkele, Russell King, Erik Mouw, and several other developers' contributions.

10 References

`http://www.powertweak.org`