Improve the Debian Boot Process
Google Summer of Code 2006

Benchmarking debian and first hotspots
Second Deliverable

student: Carlos Villegas (Carlos.Villegas at nuim.ie)
mentor: Petter Reinholdtsen (pere at hungry.com)

July 9, 2006

1 Introduction

Before trying to change the current Debian boot process, it would be convinient to analyse how the different debian releases have changed. This is done in this second deliverable for freshly installed releases of woody, sarge, etch and sid using time as metric. The results are presented at the end of Section 2.

Afterwards, in Section 4we sketch how we tested the first hotspots and compare them with the results from other people like Margarita Manterola. This would help us to realize the possibilities of time improvements and how much can they be depedent on the hardware.

Discussion on the project may be followed in the initscripts-ng-devel mailing list and the channel #pkg-sysvinit in *irc.debian.org*.

2 Benchmarking the Debian releases

Debian releases from woody have used a boot process based on the system 5 init (sysvinit) process. As it would be expected, the amount of scripts started an their order has varied with every release. In this section, we compare the boot process of the releases from woody to unstable (sid).

2.1 What is considered as boot time?

From the point of view of improving the boot process for debian, the boot time is considered to be the last stage of the boot process when the inits scripts are executed. Several unix-like system developers have managed to decrease the boot time and do other improvements like parallelization by proposing modifications to the init scripts or the way they are handled. This changes have been incorporated into some distributions like SUSE or FreeBSD[6].

On the other hand, from the point of view of the user, the boot time is considered to be the elapsed time from the moment the equipment is turned on, until it is available to be used. The status of the computer when it is available to be used may be a terminal window, an X windows manager like GNOME, or when the webserver is up.

Thus, in this report we consider two boot times as metric:

startup time - from the point of view of the user using KDE, and

sysvinit time - from the moment /sbin/init takes control of boot process.

It is obvious that sysvinit time is contained on the startup time and it is expected that the biggest variations between releases is to be in the **sysvinit time**.

The time from the moment that */sbin/init* is called, can be measured using bootchart [1] and was compared using a stopwatch. Nevertheless, as bootchart cannot see what initrd does, a small difference between the stopwatch and the bootchart can be expected. Furthermore, bootchart cannot show disk access for kernels before 2.6 as it is not available at */proc/diskstats*[5]. As a result, the bootcharts with woody and sarge don't have input-output nor disk access information on their bootchart plots.

Bootchart stops by default before KDE has finished to load as it will stop when *mingetty, agetty, rungetty* or *getty* are found. By changing the /sbin/bootchartd script for it to stop with ktip (tip of the day KDE program) we ensure that KDE is already available for the user at the moment bootchart stops logging. The changes are done in the *wait_boot* function.

2.2 Debian Woody

PC: Dell Desktop Dimension 4400 (see Appendix A) **Linux Kernel**: 2.4.16

Debian woody installs with a 2.2 kernel by default and bootchart requires a temporary filesystem (tmpfs) which is not defined for this kernel. The kernel was upgraded to 2.4 as it already provides RAM filesystem (ramfs). Besides, the /sbin/bootchartd bash script had to be modified. The changes made can be summarized as:

- kernel 2.4 was installed and the text containing *tmpfs* was changed for *ramfs* in the /sbin/bootchartd script.
- as initrd ignores the kernel call to bootchartd – it ignores the arguments given to the kernel – */sbin/init* was renamed (e.g. */sbin/init_moved*) and a link called */sbin/init* was created pointing to */sbin/bootchartd*. Finally, the bootchartd script was modified at the end such that the original init */sbin/init_moved* is called after bootchart.

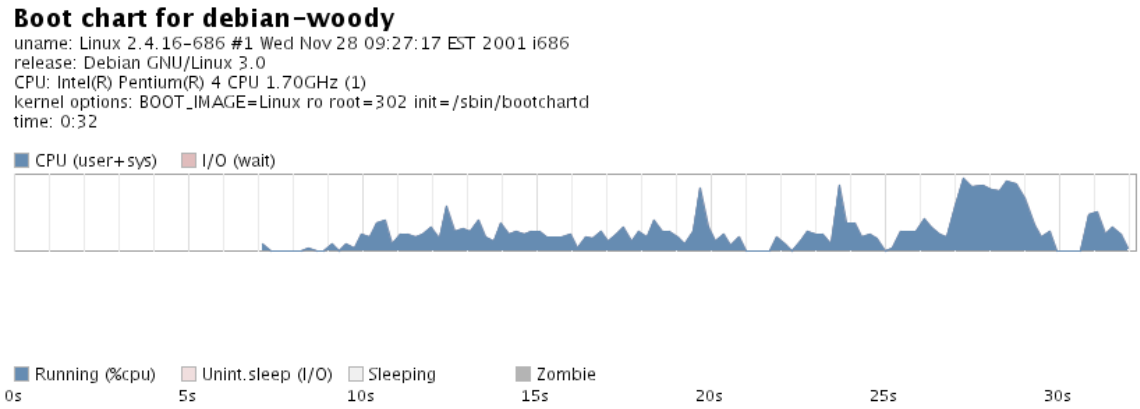

Figure 1: Bootchart for Debian woody (partial)

It may be noticed that the use of the CPU is small during the whole boot. Nevertheless, perhaps for having the smaller amount of scripts (see Appendix B), the boot is fast compared to the next release: Sarge.

2.3 Debian Sarge

PC: Dell Desktop Dimension 4400 (see Appendix A) **Linux Kernel**: 2.4.27

Used a network installation to install Debian Sarge with a desktop configuration. The windows desktop manager used is KDE.

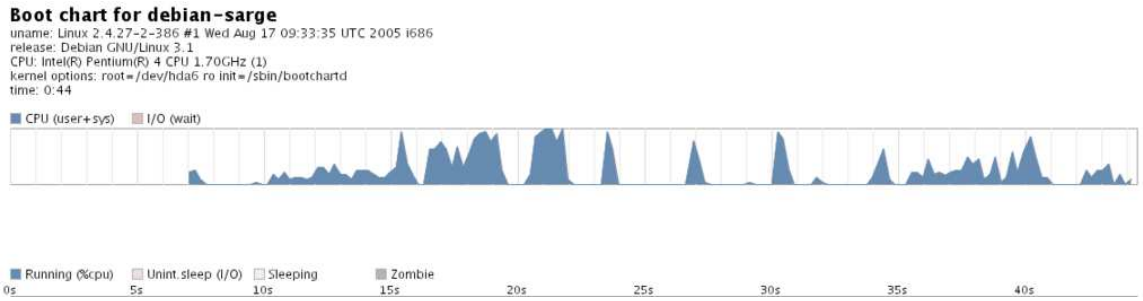

Figure 2: Bootchart for Debian sarge (partial)

With Sarge, the amount of scripts used increase considerably compared to woody (see Appendix B. With this in mind and seeing in Figure 2 the long periods of CPU inactivity it is understandable that it is the slowest release.

2.4 Debian Etch

PC: Dell Desktop Dimension 4400 (see Appendix A) **Linux Kernel**: 2.6.15

Used a network installation to install Debian Etch with a desktop configuration. The windows desktop manager used is KDE.

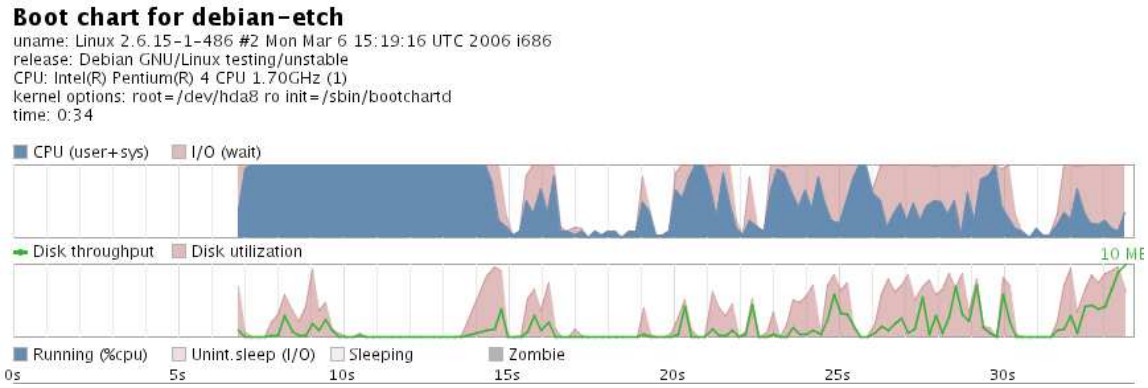

Figure 3: Bootchart for Debian etch (partial)

With Etch the use of CPU improves considerably compared to Sarge. This, together with a small increase in the amount of scripts run, should make the boot time to be smaller than sarge.

2.5 Debian Sid

PC: Dell Desktop Dimension 4400 (see Appendix A) **Linux Kernel**: 2.6.16 **sysvinit time**: **startup time**:

Distribution	**partial sysvinit time**
Woody	**32**
Sarge	**44**
Etch	**34**
Sid	**33**

Table 1: Bootchart results of fresh installed with unmodified bootchart

To install sid first a we installed Debian Etch using a network installation. Afterwards, a distribution upgrade to sid was performed followed by the installation of kdm and KDE.

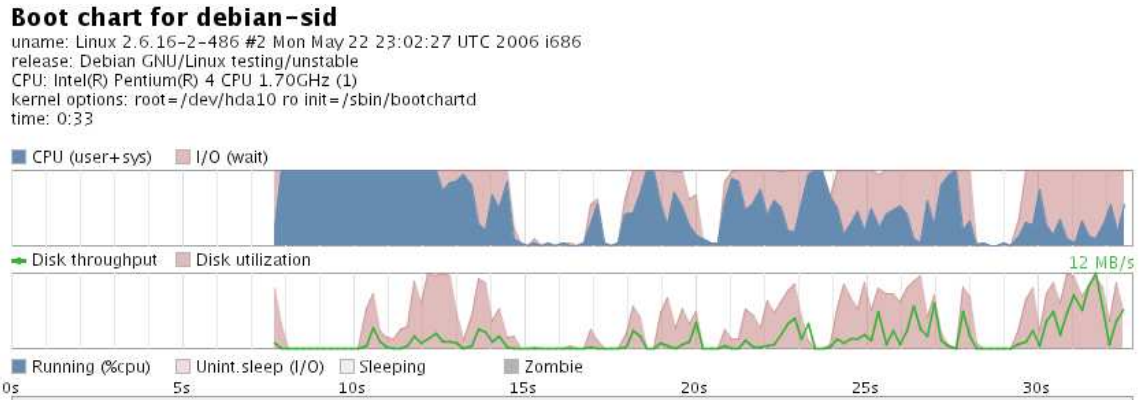

Figure 4: Bootchart for Debian woody (partial)

With the Sid release as with Etch one, the use of CPU increases compared to woody and sarge. Nevertheless, there is much time of inactivity where the resources might be better used.

3 Summary of results

Originally, using bootchart with the default configuration we obtained some partial sysvinit times depicted in the Table 1.

As we need bootchart to stop logging until KDE is usable (using autologin), we modified bootchart and obtained the following sysvinit and startup times.

Distribution	until grub	sysvinit time	startup time
Woody	11.5	59	**70.5**
Sarge	11.5	66	**77.5**
Etch	11.5	49	**60.5**
Sid	11.5	48	**59.5**

Table 2: Comparison of results

We may notice a trend towards smaller boot times from sarge to sid with an increase in the number of scripts used (see Appendix B).

Order	*Hotspot*	Marga's time[2]	our time
1	dash	6	3
2	hwclock	6	2
3	depmod	2	2
4	network	2	0
5	reorder	2	–
6	parallel	–	2
7	discover	–	0
8	preloading	–	0

Table 3: *hotspot* time reduction

4 Hotspots

Any changes resulting from this project may be implemented in Etch+. Therefore, we've used the Debian unstable release – Sid –, to test the hotspots. Recalling the discussion in Section 2.1, we use only sysvinit metric. This is based on the fact that the proposed hotspots only affect this metric.

On the other side, the computer profile used is the same as before A and the original scripts may be depicted in Appendix B

The following hotspots have been identified so far ordered by how promising they are:

4.1 dash

Use dash instead of bash reduced the boot time around 6 seconds in the tests made by Margarita Manterola[2]. This seems to be due to the smaller size of dash. The time reduction for our tests was only of 3 seconds.

4.2 hwclock

Setting up the hardware clock in the background was proposed by Margarita Manterola and she obtained a reduction of 6 seconds in the boot time[2]. Nevertheless, running hwclok on the background may cause problems with processes that can get confused if the time changes while they are running. This is the case of startpar. To overcome this problem, startpar doesn't run the scripts with sh extension in parallel.

We tried to load the hwclock on the background with a 2 seconds reduction in the boot time.

4.3 Remove depmod from the boot process

Depmod is not used any more in sid's module-init-tools and this is one of the hotspots. This was removed as a result from a discussion started by Margarita Manterola in debian-devel. Just by adding it again from an old script, the 2 second reduction from hwclock was achieved. 2 seconds were gained by Margarita Manterola[2] as well.

4.4 Set up the network in the background

Setting up the network in the background was proposed as a promising hotspot based on the results from Margarita Manterola [2] with a 2 second reduction. This hotspot was tried by modifying the networking init script but no time difference was noticed.

One problem with this hotspot is related to the network status return after running "ifup -a". In the script "ifup -a" is inside an if statement such that the status return of the network can be handled properly. This makes no sense if we send it to the background. Therefore, if this approach is widely accepted, the current init script will have to be modified.

LSB-compliance and reordering bootscripts

LSB compliance in the init scripts can be used to reorder the scripts as shown in the SUSE boot with insserv [3]. Lintian rules should be considered as well during the implementation. Just by reordering some processes around 2 seconds could be gained[2]. This hotspot has not been tested but a script to check LSB compliance was created and the ordering script of insserve is going to be modified.

4.5 Parallel execution of boot scripts

We tried parallel execution using startpar. In order to avoid unmet dependencies during the boot time, insserv was used previously to order the scripts. The small time reduction seems to be due to a deficient ordering of the init scripts by insserv as it may be observed by comparing the init scripts order before and after ordering with insserv.

In the new order of the initscripts it can be observed that:

- there are repeated versions of one initscript in the symbolic link farms (/etc/rcS.d and /etc/rc2.d) like udev,
- some scripts with dependency information (like udev and alsa-utils) were not ordered properly.

To correct this behavior requires some modifications to the code for ordering the initscripts in insserv. This would be the next step in the project (to implement a promising hotspot) just after dependency information (LSB-compliance) can be checked.

4.6 Remove discover

The discover script was removed as udev is already doing the same job. This change is valid for 2.6 kernels as older kernels don't support udev. We noticed no change on the boot time by removing discover. This should be hardware dependent.

4.7 Preloading

Having parts of the program already in memory can improve execution speed. Nevertheless, with parallel execution, preloading should be adjusted not to hinder other programs memory requirements [4] (possibly solved with dynamic preloading).

There are different programs for preloading. There is a debian package of a dynamic readahead called preload. It was tested but the default installation showed no improvement in the boot time.

4.8 Improve CPU usage with X desktop manager

Improve the use of resources when starting the desktop manager (like KDE or Gnome) like with ELF prelinking,

4.9 Make the boot less verbose

.

Making the boot print less messages to the screen should decrease the boot time. This approach probably requires to modify most scripts although the issue of deciding the information that should be showed would remain open.

4.10 Use internal functions in the init scripts

Rewritting slow shell scripts to use internal function instead of external programs is another option.

4.11 Use ELF prelinking

The use of ELF prelinking could help with the programs that need to link to many libraries, e.g. KDE.

5 Conclusions and Future Work

What explains the time difference between releases. Number of init scripts? The following steps for the project are to test the remaining hotspots and to explore briefly the interactions between them.

Besides, a script to check LSB compliance intending to be used for lintian will be created and bugs will be reported in the BTS for the scripts that don't have proper LSB headers. Finally, the script ordering of the init scripts should be improved such that dependencies are met during boot also when using parallel script execution.

Discussion on the project may be followed in the initscripts-ng-devel mailing list and the channel #pkg-sysvinit in *irc.debian.org*.

A Benchmark computer profile

Dell Desktop (Dimension 4400)

Processor: Intel(R) Pentium(R) 4 CPU 1.70GHz

CPU bus speed: 1694 MHz

Memory: 512MB

Swap: 500 MB

DISK drive: MAXTOR ATA with 80GB

B List of init scripts

References

[1] Bootchart – boot process performance visualization.
http://www.bootchart.org/.

[2] Marga's blog – parallel booting.
www.marga.com.ar/blog/index.cgi/debian/Parallel_booting.html.

[3] Suse linux reference 10.1. April 2006.
http://ftp.opensuse.org/pub/opensuse/distribution/SL-10.1/inst-source/docu/en/reference_en.pdf.

[4] Eric Brasseur. Linux optimization. May 2005.
http://www.4p8.com/eric.brasseur/linux_optimization.html.

[5] Ziga Mahkovec. Bootchart logger script.

[6] Carlos Villegas and Petter Reinholdtsen. State-of-the-art in the boot process. In *Improve the Debian Boot Process Project of Google Summer of Code 2006*, June 2006.
http://initscripts-ng.alioth.debian.org/soc2006-bootsystem/deliverable1.html.

WOODY	SARGE	ETCH	SID
rcS.d symbolic farm			
S05initrd-tools.sh	S02mountvirtfs	S01glibc.sh	S01glibc.sh
S05keymap.sh	S05bootlogd	S02mountkernfs.sh	S02mountkernfs.sh
S10checkroot.sh	S05initrd-tools.sh	S03udev	S03udev
S18hwclockfirst.sh	S05keymap.sh	S04mountdevsubfs.sh	S04mountdevsubfs.sh
S20modutils	S10checkroot.sh	S05bootlogd	S05bootlogd
S30checkfs.sh	S18hwclockfirst.sh	S05keymap.sh	S05keymap.sh
S30procps.sh	S18ifupdown-clean	S10checkroot.sh	S10checkroot.sh
S30setserial	S20module-init-tools	S18hwclockfirst.sh	S18hwclockfirst.sh
S35devpts.sh	S20modutils	S18ifupdown-clean	S18ifupdown-clean
S35mountall.sh	S30checkfs.sh	S20module-init-tools	S20module-init-tools
S39dns-clean	S30procps.sh	S20modutils	S20modutils
S39ifupdown	S35mountall.sh	S22hwclock.sh	S22hwclock.sh
S40hostname.sh	S36discover	S25libdevmapper1.02	S25libdevmapper1.02
S40networking	S36mountvirtfs	S30checkfs.sh	S30checkfs.sh
S41portmap	S38pppd-dns	S30procps.sh	S30procps.sh
S45mountnfs.sh	S39dns-clean	S35mountall.sh	S35mountall.sh
S48console-screen.sh	S39ifupdown	S36mountall-bootclean.sh	S36discover
S50hwclock.sh	S40hostname.sh	S36mtab.sh	S36mountall-bootclean.sh
S55bootmisc.sh	S40hotplug	S36udev-mtab	S36mtab.sh
S55urandom	S40networking	S38pppd-dns	S36udev-mtab
S70nviboot	S41hotplug-net	S39ifupdown	S38pppd-dns
	S43portmap	S40hostname.sh	S39ifupdown
	S45mountnfs.sh	S40networking	S40hostname.sh
	S48console-screen.sh	S43portmap	S40networking
	S50hwclock.sh	S45mountnfs.sh	S43portmap
	S55bootmisc.sh	S46mountnfs-bootclean.sh	S45mountnfs.sh
	S55urandom	S48console-screen.sh	S46mountnfs-bootclean.sh
	S70nviboot	S50alsa-utils	S48console-screen.sh
	S70xfree86-common	S55bootmisc.sh	S50alsa-utils
		S55urandom	S55bootmisc.sh
		S70nviboot	S55urandom
		S70x11-common	S70nviboot
		S75sudo	S70x11-common
		S80installation-report	S75sudo
		S99stop-bootlogd-single	S80installation-report
			S99stop-bootlogd-single
/etc/rc2.d			
S10sysklogd	S10sysklogd	S10sysklogd	
S11klogd	S11klogd	S11klogd	S10sysklogd
S14ppp	S14ppp	S14ppp	S11klogd
S19nfs-common	S18portmap	S20acpid	S14ppp
S20acct	S20dirmngr	S20dbus	S20acpid
S20exim	S20exim4	S20dirmngr	S20dbus
S20inetd	S20inetd	S20exim4	S20dirmngr
S20lpd	S20lpd	S20hotkey-setup	S20exim4
S20makedev	S20makedev	S20inetd	S20hotkey-setup
S20nfs-kernel-server	S20ssh	S20lpd	S20inetd
S20pcmcia	S20xfs	S20makedev	S20lpd
S20ssh	S20xprint	S21nfs-common	S20makedev
S20xfs	S21nfs-common	S89atd	S21nfs-common
S89atd	S89atd	S89cron	S89atd
S89cron	S89cron	S99gdm	S89cron
S99gdm	S99kdm	S99kdm	S99gdm
S99kdm	S99rmnologin	S99rmnologin	S99kdm
S99rmnologin	S99stop-bootlogd	S99stop-bootlogd	S99rmnologin
S99xdm	S99xdm		S99stop-bootlogd