HAL
open science

Package Upgrade Robustness: An Analysis for GNU/Linux® Package Management Systems

John Thomson, Andre Guerreiro, Paulo Trezentos, Jeff Johnson

To cite this version:

John Thomson, Andre Guerreiro, Paulo Trezentos, Jeff Johnson. Package Upgrade Robustness: An Analysis for GNU/Linux® Package Management Systems. 9th Open Source Software (OSS), Oct 2011, Salvador, Brazil. pp.299-306, 10.1007/978-3-642-24418-6_21 . hal-01570780

HAL Id: hal-01570780

https://inria.hal.science/hal-01570780v1

Submitted on 31 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Package Upgrade Robustness : An Analysis for GNU/Linux® Package Management Systems

John Thomson[1], Andre Guerreiro[1], Paulo Trezentos[1], and Jeff Johnson[2]

[1] Caixa Mágica Software
Edificio Espanha - Rua Soeiro Pereira Gomes
Lote 1 - 8 F, 1600-196 Lisboa {first.surname}@caixamagica.pt
[2] rpm5.org jbj@rpm5.org

Abstract. GNU/Linux systems are today used in servers, desktops, mobile and embedded devices. One of the critical operations is the installation and maintenance of software packages in the system. Currently there are no frameworks or tools for evaluating Package Management Systems (PMSs), such as RPM, in Linux and for measuring their reliability. The authors perform an analysis of the robustness of the RPM engine and discuss some of the current limitations. This article contributes to the enhancement of Software Reliability in Linux by providing a framework and testing tools under an open source license. These tools can easily be extended to other PMSs such as DEB packages or Gentoo Portage.

1 Introduction

Installation of software in Linux systems is mostly performed by installing pre-compiled binary code using a Package Management System (PMS). The most frequently used package installers are RPM Package Manager (RPM) and dpkg (Debian format). We identify methods in which package upgrades can be analysed for their reliability and to ascertain how often failures occur. By formalising the failures we hope to provide the basis for future work where failures can be classified and detected that provides a method to quantitatively assess package installers.

1.1 Background

Source code compilation on a user machine has largely been superseded by having dedicated build machines. GNU/Linux distributions then make these 'packages'[1, Sec. 3, p. 9] available as pre-compiled binary packages (Fig. 2). The MANCOOSI project[3], is dedicated to solving problems associated with various package installers and provides the most recent research in this area.

[3] http://www.mancoosi.org

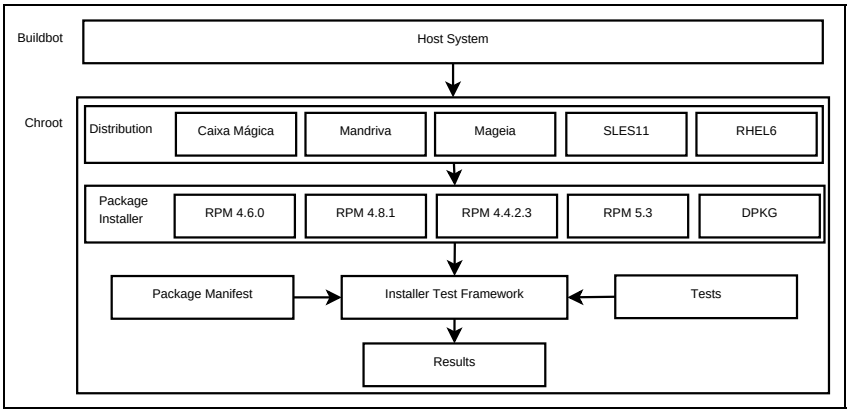

Fig. 1. An indication of the proposed architecture of the test framework, with an internal python testing system and external buildbot automation suite

There have been few investigations into package installer reliability and robustness[11, 12, 2]. Applications on GNU/Linux systems are distributed and installed through PMSs, therefore it makes sense to systematically test and analyse their reliability. RPM is the baseline PMS chosen by the Linux Standard Base (LSB) as the definitive installation system for GNU/Linux OS's[4].

There is a distinct fork of RPM Package Manager (RPM), `'@rpm5'`, referred to as RPM 5. One of the main development activities for RPM 5 is that of creating a fully Atomicity, Consistency, Isolation, Durability (ACID) compliant transactional system for installation of packages.

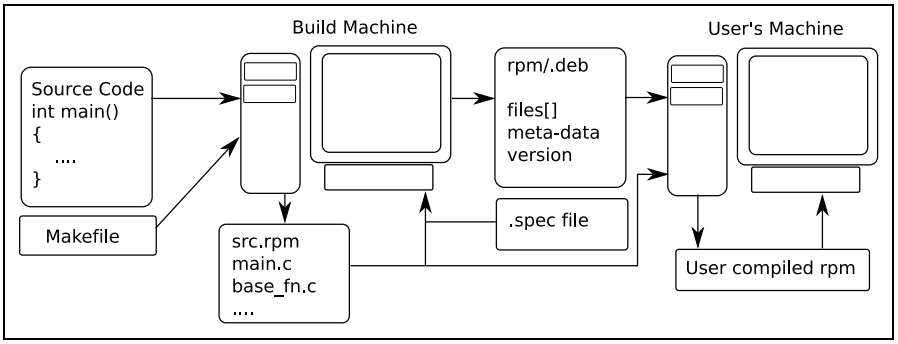

Fig. 2. A simplified topology demonstrating how sources and packages combine on a host and user machine

[4] `http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-generic/LSB-Core-generic/swinstall.html##SWINSTALL-INTRO`

One other main alternative to RPM is that of .deb packages[5], used in Debian systems and there are some subtle differences between them[1, Sec. 3.1-3.2].

Failures relating to dependency resolution which recently have been better defined [5, 7] are a matter for solvers[9][8].

1.2 The Aim

To investigate the extent that RPM systems ensure that in event of a failure that the package transaction leaves the system in a consistent state.

2 Concepts

Packages are collections of self-sufficient software that can direct PMSs. Our framework considers pre-compiled binary packages for testing.

Installers create a database of the files associated with packages and the status on the system.

An 'upgrade' is defined as when a package is modified from one version to another.

2.1 Package Upgrade Failure

Used to describe when a package upgrade operation has resulted in an erroneous installation and can occur due to a lack of physical resources and scriptlet failure (amongst others [3, Ch. 2]). Maintainer script failures remain the single biggest cause of failures.

Database Consistency Failure For RPM, there is a centralised database where package upgrades are recorded. When upgrading a package the database can fail to commit the transaction and can be left in an inconsistent state (see Sect. 3).

3 Test Framework

Software fault-injection and 'black-box' approaches are recognised methodologies for testing the robustness of systems[6, 4, 10]. The aim is to have a cross-distribution, test framework that can identify common faults and also identify unique failures, specific to architectures.

3.1 Analysable Elements

A package installer like RPM performs operations both on the file-system and on its internal database of package meta-data.

The database side of the problem is analysed since the new developments in PMSs are mainly improvements in that area.

[5] `http://www.debian.org/doc/FAQ/ch-pkg_basics.en.html`

Multiple package transactions differ from single because they have more requirements to be fulfilled: either they install all of the packages or none of them.

Within each test suite the input parameters are the Error Injection Time and the package(s) composition.

Upgrade transactions enclose two different sub-atomic operations: installation and removal. The final expected result is that only one version of a package is installed.

Failed upgrade for Individual package tests case (Sect. 4.2):

- Database consistency test: There are zero, two or more different versions present in the RPM database after an upgrade transaction;

Failed upgrade for Groups package tests case (Sect. 4.3):

- Database consistency test: Number of failures in an upgrade due to invalid database entries.
- Group Atomicity test: More than one package matches the failure case of an individual transaction.

3.2 Injecting Faults

We introduce to the normal upgrade procedure an external interruption in the form of a `SIGKILL`, signal #9 on most POSIX compliant systems, forcing a termination.

`SIGKILL` is useful for testing the robustness of RPM in a worst-case situation.

4 Test Results

Two versions of RPM are being tested with this version of the framework as can be seen in Table 1. Future versions of RPM 5 will likely be fully ACID compliant, therefore group tests should show fewer transaction failures.

4.1 Test Environment

Tests were performed on Linux Caixa Mágica (CM) 14 virtual machines with Gnome. BerkeleyDB error meant that RPM 4.6 needed a rebuild of the database after a set of interrupted package upgrades (unnecessary for RPM 5.3).

Table 1. Details of the versions of RPM being examined

Version	Type	Release	Build Date
4.6.0	Package	2.3xcm14	2009/08/03
5.3.1	Built from Source	N/A	2010/05/24

RPM 4.6.0 being tested is from a Caixa Mágica build [6].

[6] `http://contribsoft.caixamagica.pt/trac/browser/packages/cm14/rpm`

RPM 5 version used is from the RPM 5.3.1 tarball[7].

4.2 Individual Package Tests

The results shown in Table 2 are from running 100 iterations of upgrading packages with a random kill-time. Although the file-system failure rate is higher for RPM 5 the number of database failures are lower than for RPM 4.

Table 2. Individual packages transaction: File-system and database consistency failure rate after error injection

		File-System		Database	
Pkg No.	Pkg size MB	RPM 4.6 Failure Rate	RPM 5.3 Failure Rate	RPM 4.6 Failure Rate	RPM 5.3 Failure Rate
1	0.01	0/100	0/100	7/100	1/100
2	0.06	1/100	4/100	13/100	4/100
3	0.20	46/100	63/100	19/100	19/100
4	0.30	41/100	65/100	11/100	0/100
5	0.46	10/100	15/100	22/100	14/100
6	0.50	7/100	13/100	17/100	5/100
7	1.80	39/100	60/100	0/100	0/100
8	2.30	50/100	56/100	0/100	0/100
9	6.10	68/100	56/100	0/100	0/100
10	21.00	49/100	45/100	0/100	0/100
Avg.	3.27	31/100	38/100	9/100	4/100

4.3 Group Packages Tests

If a package upgrade fails in a transaction it increments the number of database failures. A single package failure in a group package upgrade transaction indicates a group transaction failure (Txn. Failure Rate). Table 3 shows the results of database consistency whereas Table 4 presents file-system consistency.

4.4 Individual Packages Against Time

If a PMS has no time to perform an upgrade there is no chance for a failure to be introduced. Figs. 3 & 4 indicate such behaviour and possibly provide an explanation for why in Table 2 the larger packages do not exhibit any database failures.

[7] `http://rpm5.org/files/rpm/rpm-5.3/rpm-5.3.1.tar.gz`

Table 3. Group transactions: Database consistency for individual packages in transaction and group atomicity failure rate after error injection

	Database Consistency		Group Atomicity	
Group (No. Pkgs)	RPM 4.6 Package Failure rate	RPM 5.3 Package Failure rate	RPM 4.6 Txn. Failure Rate	RPM 5.3 Txn. Failure Rate
1 (4)	263/400	19/400	92/100	17/100
2 (4)	189/400	45/400	83/100	36/100
3 (4)	59/400	76/400	30/100	65/100
4 (3)	11/300	31/300	10/300	31/100
5 (3)	29/300	9/300	16/100	7/100
6 (3)	30/300	39/300	18/100	53/100
7 (3)	121/300	60/300	77/100	55/100
8 (3)	73/300	65/300	73/100	65/100
9 (3)	89/300	39/300	47/100	31/100
10 (4)	55/400	1/400	55/100	1/100
Avg	92/340	38/340	50/100	36/100

Table 4. Group transactions: File-system consistency for individual packages in transaction and group atomicity failure rate after error injection

	File-system Consistency		Group Atomicity	
Group (No. Pkgs)	RPM 4.6 Package Failure rate	RPM 5.3 Package Failure rate	RPM 4.6 Txn. Failure Rate	RPM 5.3 Txn. Failure Rate
1 (4)	0/400	83/400	0/400	82/100
2 (4)	206/400	33/400	87/100	26/100
3 (4)	66/400	50/400	66/100	50/100
4 (3)	40/300	123/300	30/100	74/100
5 (3)	0/300	3/300	0/100	3/100
6 (3)	238/300	0/300	89/100	0/100
7 (3)	184/300	151/300	100/100	84/100
8 (3)	80/300	75/300	80/100	75/100
9 (3)	235/300	201/300	93/100	86/100
10 (4)	118/400	83/400	100/100	83/100
Avg	117/340	80/340	65/100	56/100

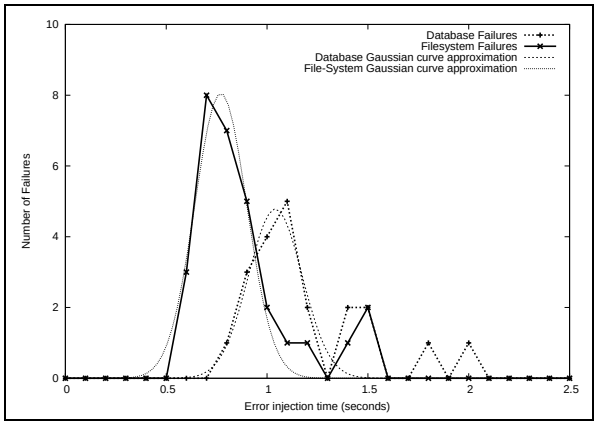

Fig. 3. RPM 4.6 - Package Failures vs. Error Injection Time for a package of 300KB over ten iterations

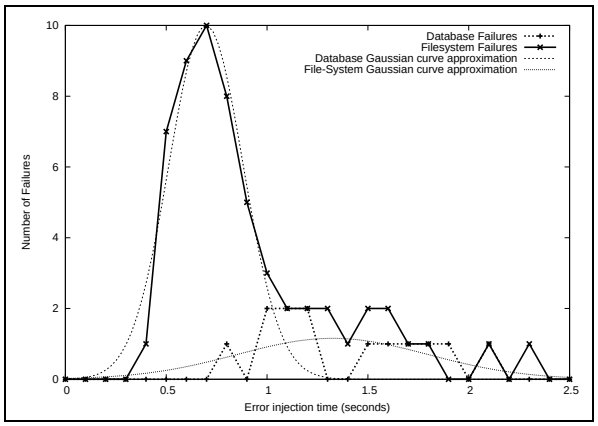

Fig. 4. RPM 5.3 - Package Failures vs. Error Injection Time for a package of 300KB over ten iterations

5 Conclusions

Linux systems are today deployed worldwide in different environments. The majority of such systems rely on a functioning kernel and a reliable Package Management System.

The `SIGKILL` test is one of the most extreme types of test likely indicating worst case behaviour. Although generally RPM 5.3 performed better in terms of database consistency, it still doesn't support atomicity in group transactions. There is the possibility to extend this study to Debian packages. Other types and permutations of errors can be injected to explore the recovery mechanisms of different PMSs.

The outcomes of this analysis, can be used to resolve problems and then suggest novel approaches for more robust package upgrade transactions.

Acknowledgements

Partially supported by the European Community's 7th Framework Programme (FP7/2007-2013), grant agreement n°214898.

References

1. Barata, P., Trezentos, P., Lynce, I., di Ruscio, D.: Survey of the state of the art technologies. Mancoosi project deliverable D3.1, Mancoosi (June 2009)
2. Crameri, O., Knezevic, N., Kostic, D., Bianchini, R., Zwaenepoel, W.: Staged deployment in mirage, an integrated software upgrade testing and distribution system. SIGOPS Oper. Syst. Rev. 41(6), 221–236 (2007)
3. Di Ruscio, D., Thomson, J., Pelliccione, P., Pierantonio, A.: First version of the DSL. Mancoosi Project deliverable D3.2, Mancoosi (Nov 2009), `http://www.mancoosi.org/reports/d3.2.pdf`
4. Duraes, J.A., Madeira, H.S.: Emulation of software faults: A field data study and a practical approach. IEEE Transactions on Software Engineering 32, 849–867 (2006)
5. Le Berre, D., Parrain., A.: On SAT technologies for dependency management and beyond. ASPL (2008)
6. Madeira, H., Costa, D., Vieira, M.: On the emulation of software faults by software fault injection. In: DSN '00. pp. 417–426. IEEE Computer Society, Washington, DC, USA (2000)
7. Mancinelli, F., Boender, J., Di Cosmo, R., Vouillon, J., Durak, B., Leroy, X., Treinen, R.: Managing the complexity of large free and open source package-based software distributions. In: ASE. pp. 199–208 (2006)
8. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: SAT'09 (2009)
9. Trezentos, P., Lynce, I., Oliveira, A.L.: Apt-pbo: solving the software dependency problem using pseudo-boolean optimization. In: ASE '10. pp. 427–436. ASE '10, ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1858996.1859087`
10. Voas, J.: Fault injection for the masses. Computer 30(12), 129–130 (1997)

11. Yoon, I.C., Sussman, A., Memon, A., Porter, A.: Effective and scalable software compatibility testing. In: ISSTA '08. pp. 63–74. ACM, New York, NY, USA (2008)
12. Zacchiroli, S., Cosmo, R.D., Trezentos, P.: Package upgrades in foss distributions: Details and challenges. In: First ACM Workshop on HotSWUp (October 2008)