ANALYSIS OF PERFORMANCE AND DESIGN PROBLEMS OF RPM DATABASE

Jan Zelený
Master Degree Programme (2)
E-mail: xzelen11@stud.fit.vutbr.cz

Supervised by: Petr Peringer
E-mail: peringer@fit.vutbr.cz

ABSTRACT

This paper focuses on RPM packaging system and the concepts it uses to manipulate with its local metadata storage (RPM database). It analyzes current solution and outlines some possible upgrades – whether they are performance related or they represent new features. To provide better view, comparison with some similar systems is included as well.

1 INTRODUCTION

Package management systems are very effective solution for managing software in a computer, mainly used in various Linux distributions. Their basic idea and goal is that *all information about installed software is kept on one place and is managed by common utitilities*. PM systems are composed of 2 basic parts – metadata storage and tools performing desired operations.

RPM is one of mostly spread PM systems. These days it is much more complex than it used to be when it started. Because of this, it is important to upgrade all parts, so the whole system can keep up with rising demands of its users. This paper will analyze current solution of RPM database and suggest its possible upgrades.

2 CURRENT RPM

RPM is composed of several programs. They all use *rpmlib*, the basic library for all operations with RPM packages.[1] One of its subsystems handles database operations. The database runs on Berkeley DB backend. This very flexible and general-purpose engine is based on a simple concept: every database stores values as byte strings, no data types are defined or known by the database. Data values are internally ordered by index values, which are byte strings as well.

The RPM database consists of several files. *Packages* file contains the primary database. As values, complete headers of RPM packages are stored. Ordinal numbers are used as their index. Other important files contain secondary databases, which are used as indices for the primary one. They are all based on the same pattern. As an example of this pattern: database indexing package names has these names as an index and as value a list of all packages having particular name is stored. In this list packages are represented by their number from primary database.

To have an image about performance shortcomings, it is important to run some profiling tests. These test were run on a database about 400 MiB large. This database contains records about all packages that are available in Fedora repository. First install these packages, then run some queries on them and then uninstall them. All commands are measured with profiling tool (callgrind or gprof). The command set is as follows:

```
cd /path/to/temp-db/directory
rpm -i --justdb --nofiles --nodeps --dbpath `pwd` <packages>
rpm -q --info --dbpath `pwd` rpm
rpm -qa --dbpath `pwd`
rpm -qa --qf "[%{FILENAMES}\n]" --dbpath `pwd`
rpm -e --nodeps --justdb --dbpath `pwd` <packages-in-db>
```

Bottleneck during installation is indexing of files and directories. They are added to a *list of available features* and this list is sorted for each installed package. BDB related operations take about 10% of CPU time. This can't be reduced much, all package information has to be stored in the database. The only possible reduction here can be done by reducing the ammount of data stored in the database. More interesting is conversion of package lists (in sec. databases) from stored to working format, it takes about 17.5% of CPU time. The main issue is byte order setting and cardinality of tuples stored in the list.

What takes erasing operations a long time is cleaning of package lists. It has to be done for each secondary index. Erasing consists of quicksorting and subsequent binary searching in set. These two functions take most of the CPU time during erasing operations.

Besides hash checking which can be disabled, the largest time consumer during queries is header parsing (around 10% of total time) in form of `headerLoad` function. The second largest (database related) is loading data from rpmdb. As in previous cases loading the whole header is the most significant performance retarder related to rpmdb. It is responsible for both above mentioned time consumers.

The last and the most significant performance retarter is file syncing. Databases are synced after every operation. That is without a doubt good for robustness, but it takes a long time. With large amount of operations, this time is unacceptable. Now to the non-performance issues.

The main one might be a size of the database. Optimizing this would mean go against performance optimization, hence it is important to balance these two. What can be measured (and maybe reduced without performance penalty) is database overhead. Measurements were taken on database composed of all package installed in Fedora 9, 10, 11 and 12. Then a small database (just about $2 MiB$) has been created. By simply calculating a slope of line between overhead of the smallest and the largest database an overhead of 16% was determined.

3 COMPARISON WITH OTHER PM SYSTEMS

There are several other PM system, which have interesting features worth comparing with RPM. The first one is **Conary**, used in ForesightLinux and developed by rPath Inc. The system is written in python and can utilize several SQL engines as backends. [3] It has some features giving him a degree of advantage over RPM. The most interesting are unification of a local database handling with repository related work and replacing scriptlets, since they represent a major obstacle for package portability between distributions.

Another interesting program is *yum* and its concept where the database is divided to more parts, so we don't need to load all package metadata at once. *pisi*, another PM system, uses BDB as well. It brings repository and package handling closer and its database has more parts. Mentionable concept is that pisi stores information about all the packages in registered repositories, along with their state in local system. [5]

Most of packaging systems use plain text files to store package information. This approach has major advantage – human readability and extensibility. This advantage however can have its price and that is performance. This applies for systems like *dpkg*, which stores list of all installed in one large file. On the opposite side, there is *ABS*, used in Arch Linux. The format is very similar to that used by *portage*, in Gentoo and *ports* in BSD. [4] It uses directory tree to store information about installed packages. In this tree, every package has its own directory, which contains all of packages' stored metadata. This approach can perform better than database, because all indexing and similar problems are solved by filesystem.

One differences between RPM and almost every other PM system are dependencies. Other PM systems use package-level dependencies, RPM uses feature and file focused dependencies. Thus RPM generates more data which have to be stored in the database.

4 CONCLUSION

This paper analyzed current RPM database and compared it with some similar solutions. Based on this paper some modifications to the database can be suggested to improve it. Here is a basic summary of them. First of all, the amount of data loaded from database should be reduced significantly. This can be achieved by several ways – reducing data in package headers, splitting the database or using relational database as backend (relational data model defines that data has to be atomic, not in a blob). Then some other performance bottlenecks were identified, which should be easy to solve. Some features were also suggested in order to promote RPM: enclosing repository and local database, re-think the concept of scriptlets and change dependency level.

ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant FIT-S-10-1 and the research plan MSM0021630528.

REFERENCES

[1] Troan E., Ewing M., RPM team: Source code of RPM 4.7.0, 2009

[2] Oracle: Getting started with Berkeley DB <*http://www.oracle.com*>

[3] rSync Inc.: Conary API documentation, <*http://cvs.rpath.com/conary-docs/*>

[4] Archlinux – Arch Build System: <*http://wiki.archlinux.org/index.php/Arch_Build_System*>

[5] Pisi database schema *http://svn.pardus.org.tr/uludag/trunk/pisi/doc/pisi-db.xmi*

[6] Vidal S., Macken L., Antill J.: Source code of createrepo utility