Using `apt-get` With External Solvers

Pietro Abate, Roberto Di Cosmo, Ralf Treinen, Stefano Zacchiroli

May 9, 2020

Contents

1 Introduction

`apt-cudf` bridges `apt-get` with external CUDF solvers. It implements a translator from the EDSP [] format to the CUDF [] format, invokes an external solver, and transmits back to `apt-get` the outcome using the EDSP format.

This primer applies to version 5.0.1 of `apt-cudf`.

2 Why Using External Solvers with `apt-get`

3 Basic Usage

Starting from release 0.9.x, `apt-get` is able to use external solvers via the EDSP protocol. In order to use `apt-get` with an external solver you need to have installed , besides `apt` itself, the package `apt-cudf` and at least one solver package. Currently available solver packages in debian are `aspcud`, `mccs`, and `packup`.

The integration of CUDF solvers in `apt-get` is transparent from the user's perspective. To invoke an external solver you just have to pass the option `--solver` to `apt-get`, followed by the name of the CUDF solver to use. These solvers use different technologies and can provide slightly different solutions.

Using an external CUDF solver does not require any other particular action from the user. The `--simulate` (or `-s`) option is used here to make `apt-get` just display the action it would perform, without actually performing it:

```
$apt-get --simulate --solver aspcud install gnome
NOTE: This is only a simulation!
      apt-get needs root privileges for real execution.
      Keep also in mind that locking is deactivated,
      so don't depend on the relevance to the real current situation!
Reading package lists... Done
Building dependency tree
Reading state information... Done
Execute external solver... Done
The following extra packages will be installed:
[...]
```

Depending on the solver, the invocation of an external solver can take longer then the `apt-get` internal solver. This difference is to due to the additional conversion step from EDSP to CUDF and back, plus the effective solving time.

`apt-get` itself ships two EDSP-compatible tools:

1. `internal` (since release 0.8.x) refers to the internal `apt-get` dependency solver;
2. `dump` is not a real solver but just dumps the EDSP document into the text file `/tmp/dump.edsp`.

For example, the following invocation is equivalent to invoking `apt-get` without the `--solver` argument:

```
apt-get install --solver internal <package-to-be-installed>
```

4 Advanced Usage

CUDF-based solvers understand user preferences and use them to select a best solution. Compared with `apt-get`, this gives the user a greater flexibility to define "optimal" solutions for installation problems, for instance to minimize the number of new packages that are installed, or to minimize the total installation size the packages to upgrade.

4.1 Optimization Criteria

4.1.1 Using Optimization Criteria

Each CUDF solver understands a basic optimization language, and some of them implement extensions to this basic language to implement more sophisticated optimization criteria. `apt-cudf`, that is the bridge between `apt-get` and the CUDF solver, associates to each `apt-get` command an optimization criterion that can be either configured at each invocation using one `apt-get` configuration option or by using the configuration file (`/etc/apt-cudf.conf`) of `apt-cudf` (see Section 4.1.2 for their precise meaning):

```
solver: *
upgrade: -count(new),-count(removed),-count(notuptodate)
dist-upgrade: -count(notuptodate),-count(new)
install: -count(removed),-count(changed)
remove: -count(removed),-count(changed)
trendy: -count(removed),-count(notuptodate),-count(unsat_recommends),-count(new)
paranoid: -count(removed),-count(changed)
```

The field `solver` defines the (comma-separated) list of solvers to which this stanza applies. The symbol "*" indicates that this stanza applies to all solvers that do not have a specific stanza.

Each field of the stanza defines the default optimization criterion. If one field name coincides with a standard apt-get action, like `install`, `remove`, `upgrade` or `dist-upgrade`, then the corresponding criterion will be used by the external solver. Otherwise, the field is interpreted as a short-cut definition that can be used on the `apt-get` command line.

Using the configuration option of `apt-get` `APT::Solver::aspcud::Preferences`, the user can pass a specific optimization criterion on the command line overwriting the default. For example :

```
apt-get -s --solver aspcud install totem -o "APT::Solver::aspcud::Preferences=trendy"
```

4.1.2 Defining Optimization Criteria

Sets of packages The measurements that may be used in optimization criteria are taken on selected sets of packages in order to measure the quality of a proposed solution. In this context, when we speak of *package* we mean a package in a specific version. That is, a package with name p and version 1 is considered a different package then the one with the same name p and different version 2. We will denote with I the set of packages (always name *and* version) that are initially in state *installed* on the machine, and with S the set of packages that are in state *installed* as a result of the `apt-get` action.

solution the set S

changed the symmetric difference between I and S, that is the set of packages that are either in I and not in S, or in S and not in I.

new the set of packages in S for which no package with the same name is in I.

removed the set of packages in I for which no package with the same name is in S.

up the set of of packages in S for which a package with the same name but smaller version is in I.

down the set of of packages in S for which a package with the same name but greater version is in I.

Measurements on sets of packages Several ways to measure sets are defined. All these measurements yield an integer value. Here, X can be any of the sets defined above:

count(X) the number of elements of set X

sum(X,f) where f is an integer package property. Yields the sum of all f-values of all the packages in X.

Example: sum(solution,Installed-Size) is the size taken up by all installed packages when the `apt-get` action has succeeded (as declared in the Packages file).

notuptodate(X) the number of packages in X whose version is not the latest version.

Example: notuptodate(solution) is the number of packages that will be installed when the `apt-get` action has succeeded, but not in their latest version.

unsat_recommends(X) this is the number of recommended packages in X that are not in S (or not satisfied in S, in case the recommendation uses alternatives).

For instance, if package a recommends `b, c|d|e, e|f|g, b|g, h` and if S is $\{a, e, f, h\}$ then one would obtain for the package a alone a value of 2 for unsat_recommends since the 2nd, 3rd and 5th disjunct of the recommendation are satisfied, and the 1st and 4th disjunct are not. If no other package in X contains recommendations that means that, in that case, unsat_recommends(X)=2.

aligned(X,f1,f2) where $f1$ and $f2$ are integer or string properties. This is the number of of different pairs $(x.f1, x.f2)$ for packages x in X, minus the number of different values $x.f1$ for packages x in X.

In other words, we cluster the packages in X according to their values at the properties f1 and f2 and count the number of clusters, yielding a value v1. Then we do the same when clustering only by the property g1, yielding a value v2. The value returned is v1-v2.

Combining Measurements into Criteria An optimization criterion is a comma-separated list of signed measurements.

A measurement signed with + means that we seek to maximize this value, a measurement signed with − that we seek to minimize this value. To compare two possible solutions we use the signed measurements from left to right. If both measurements yield the same value on both solutions then we continue with the next signed measurement (or conclude that both solutions are equally good in case we are at the end of the list). If the measurements are different on both solutions then we use this measurement to decide which of the solutions is the better one.

Example 1: `-count(removed), -count(changed)`, sometimes called the *paranoid* criterion. It means that we seek to remove as few packages as possible. If there are several solutions with the same number of packages to remove then we chose the one which changes the least number of packages.

Example 2: `-count(removed),-count(notuptodate),-count(unsat_recommends),-count(new)`, sometimes called the *trendy* criterion. Here we use the following priority list of criteria:

1. remove as few packages as possible
2. have as few packages as possible in a version which is not the latest version
3. have as few as possible recommendations of packages that are not satisfied
4. install as few new packages as possible.

4.2 Pinning

4.2.1 Strict Pinning and Its Limitations

When a package is available in more than one version, `apt-get` uses a mechanism known as pinning to decide which version should be installed. However, since this mechanism determines early in the process which package versions must be considered and which package versions should be ignored, it has also the consequence of considerably limiting the search space. This might lead to `apt-get` with its internal solver not finding a solution even if one might exist when all packages are considered.

Anther consequence of the strict pinning policy of `apt-get` is that if a package is specified on the command line with version or suite annotations, overwriting the pinning strategy for this package, but not for its dependencies, then the solver might not be able to find a solution because not all packages are available.

4.2.2 Ignoring Pinning

To circumvent this restriction and to allow the underlying solver to explore the entire search space, `apt-get` can be configured to let the CUDF solver ignore the pinning annotation.

The option `APT::Solver::Strict-Pinning`, when used in conjunction with an external solver, tells `apt-get` to ignore pinning information when solving dependencies. This may allow the external solver to find a solution that is not found by the `apt-get` internal solver.

4.2.3 Relaxed Pinning

Without relaxing the way that pinning information are encoded, `apt-cudf` with an external CUDF solver would be effectively unable to do better then `apt-get` because important information is lost on the way. In order to overcome this limitation, `apt-cudf` has the ability to reconstruct the user request and to use this information to provide a possible solution. To this end, `apt-cudf` reads an environment variable, named `APT_GET_CUDF_CMDLINE` which the user can pass along containing the invocation of `apt-get`.

To make it straightforward for the user, a very simple script called `apt-cudf-get` is provided by the `apt-cudf` package.

```
#!/bin/sh
export APT_GET_CUDF_CMDLINE="apt-get $* -o APT::Solver::Strict-Pinning=\"false\""
apt-get $* -o APT::Solver::Strict-Pinning="false"
```

The wrapper is invoked using the same commands as `apt-get`:

```
apt-cudf-get -s --solver aspcud install totem \
    -o "APT::Solver::aspcud::Preferences=-count(new),-count(changed)"
```