Findings from my first foray into PCP - updating the lmsensors metrics (PMDA)

March 2019

Christian Horn / chorn@redhat.com
AMC Technical Account Manager
Red Hat CEE

Agenda

- /whois Christian Horn
- The motivations for making the changes
- What should pmda-lmsensors ideally look like?
- Implementation
  - Choosing the language to implement
  - How to read the sensors?
  - QA-tests for pmda-lmsensors
- The various implementation stages, documentation, aids for new developers and ideas for further improvement
- Community interactions

/whois Christian Horn

- born in Mühlhausen/Thüringen/East germany
- lived 15 years in Munich
  - #linuxadmin #engineer #architect #typography
  - First monitoring for webfarm: Perl, rrdtool, apache
- since 3 years in Tokyo
  - loving #culture #people #language #food
- working as: Technical Account Manager @Red Hat
  - #debugging #coding #documenting #fixing-things

Motivation

- Summer 2018: Investigated an introduction article into PCP (in the meantime published at <link>)
- Goal: illustrate how to use pmdas (agents to read metrics): pmda-postgresql, pmda-apache and so on
- pmda-lmsensors sounded like a lightweight pmda to read sensor data - just right for demonstrating how simple pmdas can be used!

- Expectation at that time:
  - install the pmda, run Install, have metrics appear, end of story.
- Reality:
  - pmda-lmsensors handled just a hand full of hardware sensors, not bringing up any metrics on my systems

So, how hard could it be to get _my_ sensors monitored?

What should pmda-lmsensors look like?

I want to just run

```
$ dnf install -y pcp-pmda-lmsensors lm_sensors
$ cd /var/lib/pcp/pmdas/lmsensors
$ ./Install
```

and then see the metrics tree polulated:

```
$ pminfo -f lmsensors
lmsensors.thinkpad_isa.fan1
     value 0
lmsensors.coretemp_isa.temp3
    value 40
lmsensors.coretemp_isa.temp2
    value 39
[…]
```

..and that’s what the final implementation in PCP 4.2+ does now.

Choosing the language, 1of2

- Canonical choices for pmda’s: Python, Perl, C
- Python?
  - Pro: Best reputation for maintainability, high abstraction layer, low amount of code to write
  - Con: might not be best for performance critical pmda’s
- C?
  - Pro: fast
  - Con: much code to write and maintain
- Perl?
  - Pro: high abstraction layer
  - Con: maintainability.., and might not be best for performance

Choosing the language, 2of2

- Lmsensors rewrite initial language: Perl
- Because:
  - I had prior knowledge
  - I had zero knowledge with Python, and not much with C
- Prior code: My old Thinkpad x200 had a bad firmware which never tuned down the fan once it did spin up. I ‘fixed’ that with a Perl daemon monitoring the sensors and manually tuning down the fan.

How to read sensors most efficiently?

- ‘lmsensors’ project: binaries, but no abstracted library to use.
- First code in Perl: parsing ‘sensors -u' output, just to get fast results instead of manually looking at /sys-fs files. 'sensors -u' gets executed once for each readout of the lmsensors metrics
- Second implementation in Perl: reading /sys-fs directly, to compare performance with former approach. Should be faster as the files are accessed directly..
- implementing code which compares 1000 runs of /sysfs-read vs. executing 'sensors -u' - turns out its comparable.
  - Memory consumption also comparable.
  - comparing both approaches, ‘sensors’ has the bonus of allowing each sensors flexible name modification (temp1 -> cpu core1), and value fixing (multiply the reported value with 2.5 before outputting).

First pull request, learning about build check

- Got first implementation in Perl together, did run it for some time on my box: let's send a pull request.
- Sidetracked into getting the required git commands together
- Eventually: https://github.com/performancecopilot/pcp/pull/559
- I realize that automatic build checking is done!
  - Log of that directly readable.. \o/
  - and the build check fails <o>
- **Expectation:** HEAD (so far committed/accepted code) should always be clean/pass all the tests. So my code is causing an issue then.
- So.. a weekend of staring at the code, I notice in the logs from above that tests for pmda-bcc failing, which I did not touch at all?
- **Reality:** HEAD is not always clean, it failed also without my code. Good to know.. ;)

Second pull request

- Second request: https://github.com/performancecopilot/pcp/pull/564
- Passing the automatic tests, getting a person to check (Thanks, Mark!)
  - copyright notes in the new files missing
  - We should ensure binary 'sensors' is available, but how?
    - dependency in specfile: just good for rpm based distros
    - binary check at pmda start? More generic.
    - Going with both.
  - No QA tests for the new code exist.
- Hm.. what are these QA tests anyway?
- PCP is around quite some time, many platforms. Linux distros, BSD, Unix etc. QA scripts are used to notice when things break: Is the cpu load counter outputting useful values, on various platforms? Is pmda-postgresql still useful after last changes? etc.

QA tests for pmda-lmsensors

- All systems will have different sensors - some none at all: virtual guests.
- Lets first check if the sensors binary is available for the QA script.
  - How many sensors does 'sensors -u' report?
  - Does 'pmrep lmsensors' reporting the same number?
- What can we do to verify the reported sensor values? Not trivial. We could run 'sensors -u' and 'pmrep lmsensors', and compare their metrics, but have some serious issues:
  - a) The order between both can be different. If ‘sensors -u’ outputs the CPU temperature is first, but pmrep last, then we would first need to do reordering of entries.. that is error prone.
  - b) We measure values which have tendency to constantly change. Measuring at 2 times likely to produce different results.
  - c) sensor values and fan values are reporting metrics in completely different scales: tens vs. thousands.
- Solution for now: computing the average of all sensor values from both sources, and checking if results differ more than 10%. Might break if a fan starts/stops.. but I guess when running the suite the fan runs anyway.

Next pull request, 1of2

- Now I got a useful QA test script together, and pmda-lmsensors also came into shape. Next pull request.
- The PCP-team commented 'we take the Perl rewrite, but would have preferred Python'
- Makes me wonder: would Python do better or worse than Perl, considering resources?
- Just one way to find out: rewrite in Python, my first Python code
- Comparing Perl/Python:
  - Turns out: memory requirements w/ Perl: 10MB, Python3: 18MB!
    - => Probably not relevant for a user.
  - But then there is the code maintainability.. so now that the Python version is written, I did send the pull request for that

Next pull request, 2of2

- Good news: the QA test script is just a wrapper which does end-to-end checking, so independent of the language implementing the pmda, so no need to change the already written QA tests from Perl version :)
- Pull request: https://github.com/performancecopilot/pcp/pull/567
- That code
  - looked good on my 2 Thinkpads for some weeks
  - Passed the QA test scripts
  - passed the automatic pull request checks
- ..and got merged into upstream \o/

First bug reports.. debugging 1of3

- First user reports from the team trying out the new code.
- Getting 'sensors -u' output for debugging. Good that I am not reading /sys directly ;)
- How to debug a Python pmda? Multiple issues.
- The pmda code is designed to be run from PMCD, not from the command line. In that mode, one can not see direct debugging output. Printing to STDOUT lands in PMCD.

```
[root@gagarin lmsensors]# ./pmdalmsensors.python
p
 &�p
    &�[root@gagarin lmsensors]#
[root@gagarin lmsensors]#
```

- So my first approach was to implement own logging into files.. that is just a hack.

Additional hint: dbpmda(3) should be considered to debug pmdas

Debugging 2of3

- Implemented debug levels with 'import argparse':

```
[...]
parser.add_argument("-d", "--debug", type=int, choices=[0, 1, 2],
          help="change debug level, 0 is default")
[...]
```

- Call like this:

```
[root@gagarin lmsensors]# cd /var/lib/pcp/pmdas/lmsensors
[root@gagarin lmsensors]# ./pmdalmsensors.python -d2
```

- This will tell pmda-lmsensors “Hey, you are not under PMCD control, but in debug mode. Please write to STDOUT.”
- Shows details of sensors parsing and more. Now we have a way to debug.

Debugging 3of3

- Next debug issue: I have just my 2 thinkpads and KVM guests, yet need to investigate bug reports from systems with completely different sensors
- Implementing

```
parser.add_argument("-i", "--inject", type=argparse.FileType('r'),
        help="inject data from file instead of using sensors")
```

  for side loading/injection of external sensor values
- Now I could rewrite the parser, to understand the cases where the old implementation had issues.
- Small collection of 'sensors -u' files stacks up.
- Pull request https://github.com/performancecopilot/pcp/pull/570 reworks the parser, makes all sensor outputs I have seen happy.
- PCP release 4.2 includes the new code. \o/

Room for improvement

- QA tests could be extended to use injection and verification of expected outcome with some example ‘sensors -u’ outputs
- Each read of the metrics runs 'sensors -u'. Run multiple instances of 'pmrep lmsensors' in parallel, each one triggering one execution per second with default time frame; things sum up. If common use case. one could implement a caching mode to be used by default:
  - the pmda could just _once_ per second read the values and answer queries out of cache mostly.
- Somehow detect in the pmda-code whether called from PMCD or plainly. The pmda could automatically enter debug mode then.
- ‘gkrellm’ pointed me at new sources for sensors which lmsensors does not capture. Should these be implemented directly in pmda-lmsensors or via lmsensors? Upstream lmsensors should have an opinion?

Summary

- Contributing is not so hard at all.
- Upstream is not expecting perfection. In exchange, you should commit to spend some time to the contribution. It's also not done with your pull request getting accepted, but with people using it and reporting.
- I might in my role as TAM get customer issues regarding my own code onto my desk ^^
- It's a good feeling to see ones own code distributed - and that is encouraging you put some effort into it. This also helped me to better understand engineering’s views. I am usually in the role of requesting fixes for customers, not implementing them.

Community interactions

- mailing list pcp@groups.io – best for asking and discussing asynchronously
- Github.com – issues discussion, also async
- #pcp on Freenode – IRC is the well proven protocol for live interactions, if someone is around on the channel
- Hackfests, dedicated time @confs – might be an effective way to get fast feedback. I guess many of the points taking time for me would have been solved quicker

How do <u>you</u> interact with the community in the projects you are involved in?

Community interactions

- Further keywords:
  - github PR/Issue templates
  - auto assigning reviewers to PR's via a bot
  - QA/CI on each request
  - mergebots
  - What else is expected in an upstream community?
    - IRC? Slack?
    - Mailing List used much anymore?

potential improvements

This was gathered in our feedback session at the conference.

- 1. Show more best practices
  - Consider the end-user (sys admin / devops) view
  - how to configure things
  - how to install optional PMDAs
  - Create chef|puppet|ansible recipes or playbooks
- 2. Newbie developer guidelines
- 3. PMDA developer guidelines
  - PMNS
  - Metadata
  - Callbacks
  - instance domain rules and management
  - dbpmda

Thank you!

どうもありがとうございました！
Спасибо!
Danke!

Christian Horn / chorn@redhat.com
Technical Account Manager
Red Hat CEE

Backups: nice to have

- Seeing pmda-lmsensors unusable for me, I actually thought: PCP is likely prepared to offer simple monitoring of custom application metrics. Either
  - reading a number of a text file on the file system
  - or execution of a batch file/command which outputs the current metric to STDOUT?
- $(sensors -u | grep temp1_input | sed -e 's,.*: ,,') is all I need!
- Reality: no simple wrapper-pmda seen for this. Next best thing is to take a simple pmda, and implement own metrics. Requires Perl, Python or C usage, instead of calling 'my' batch file.
- So second PCP blog article idea was born: how to do a simple own monitoring of a single sensor value. Without explaining to much background, just right article length for the audience (public by now)

Brainstorming/requests

- Which functionality are you missing in PCP, what would you like to see?
- What is the background/why do you want to see it?

- Maybe someone else with free cycles is looking for an idea!
- Maybe the idea just needs further input to get realized?

Idea pmda-denki (electrical power), 1of3

- idea was born here: https://github.com/performancecopilot/pcp/issues/532
- Basically:
  - Provide a metrics on power consumption, provide as metric what powertop provides for the moment
- Why? For example because of climate change. I see this in Europe more than in America or Japan, for many decisions people start to consider influence on the environment. Pmda-denki could allow pull request testbuilds coming back with “your code is 10% faster than what we had previously, but consumes 30% more power.”
- "This nginx did run for a week. How long would I need to run this bicycle power generator to generate the same energy?"
- Compare power-efficiency of GPU and CPU for a certain calculation

Idea pmda-denki (electrical power), 2of3

- Prior code/approaches:
  - Could we benefit from what Android is doing there?
  - Intel powertop <link> is big, not offering functions via a library
  - PowerAPI <link> looks like an approach
  - Powerstat <link> is maybe the best code to start with, one could use that code as base. Directly in a pmda or abstracted as a library, and then be used from a library. Maybe besides library also a constantly running user space daemon would be required.
  - Query power via IPMI to remote service boards of servers
- The github thread <link> has more ideas

Idea pmda-denki (electrical power), 3of3

- Challenges:
  - We might not have a direct sensor for consumption, but we know
    - how busy the system is
    - we can read temperature sensors
- What is electric energy converted into? Electric and magnetic fields, and thermic energy (heat)
- Depends on surrounging temperature of course, but if we can measure that, we can look at how much we heat up.