Significant performance improvements

Jaroslav Mracek
Principal Software Engineer
Red Hat

What is significant?

Example no. 1

Code is 100 times faster. Is it significant?

What is significant?

Example no. 1

Code is 100 times faster. Is it significant?

- Originally it required 1 ms from 10 s application run
- Used rarely
- No significant difference for users

What is significant?

Example no. 2

Code is 50% faster. Is it significant?

What is significant?

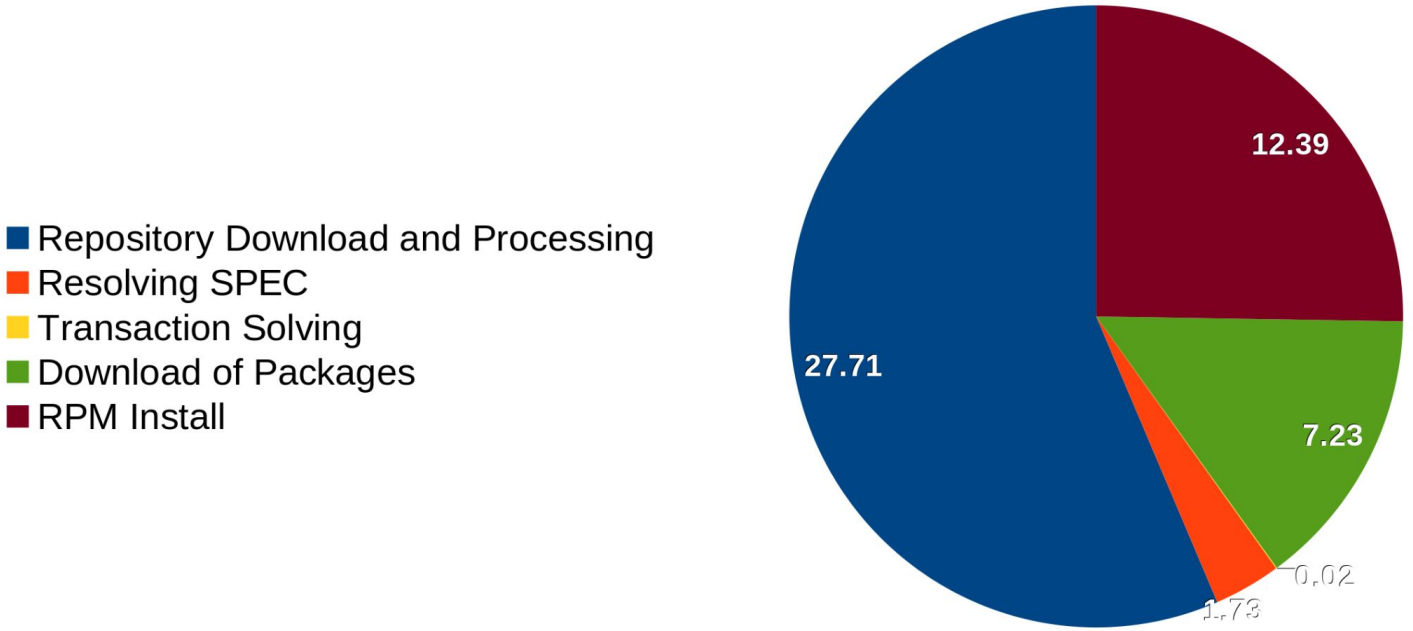
Example no. 2

Code is 50% faster. Is it significant?

- Reduce application load time from 10 to 5 seconds

Potential for Improvements

dnf install dnf

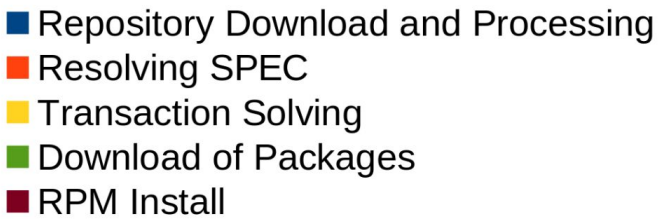


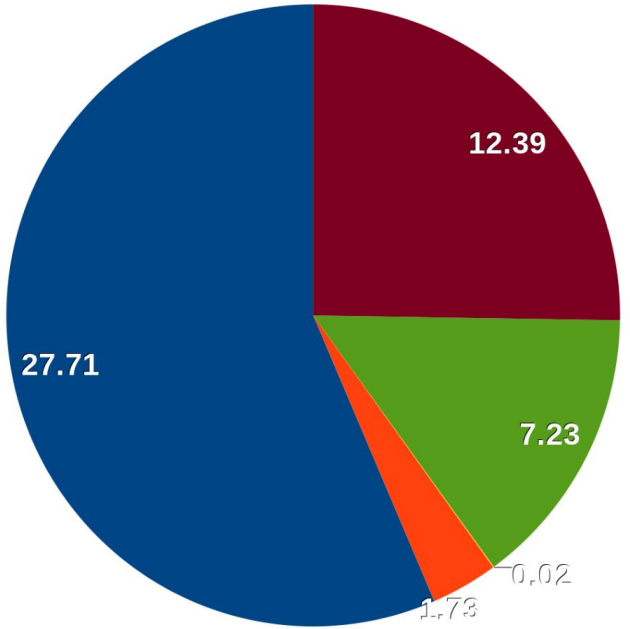
Performance improvement - Loading of repositories

Resources

Potential for Improvements

dnf install dnf





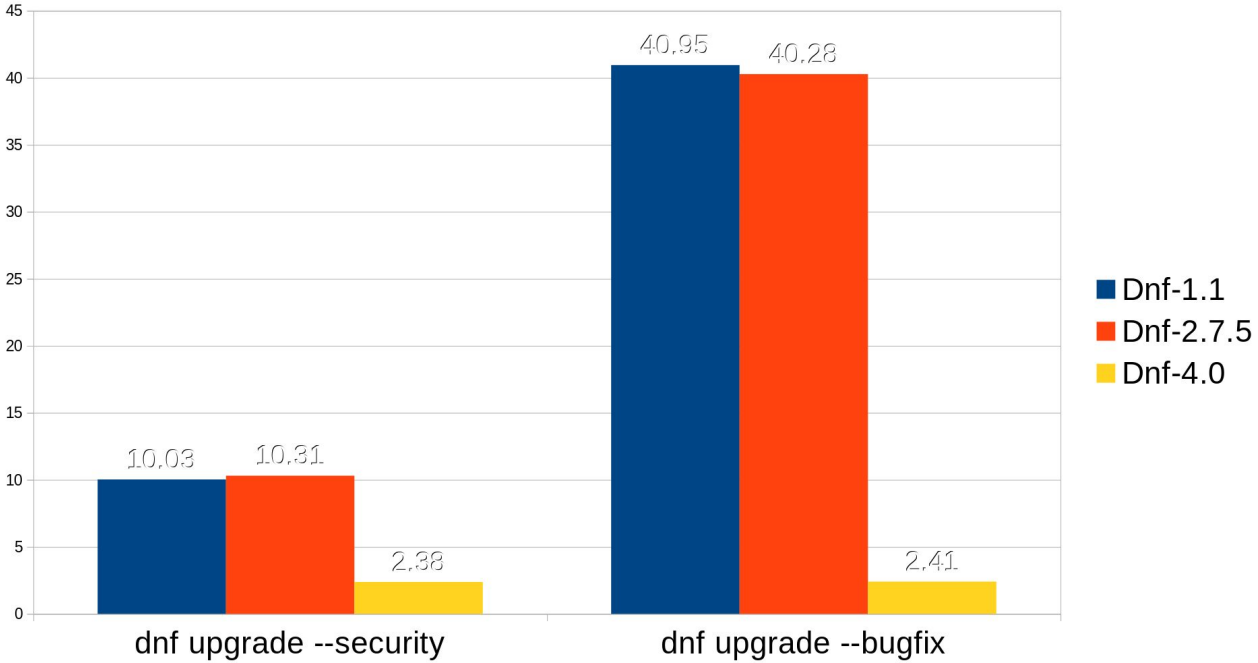
Potential for Improvements

- User reports
  - Bugzilla
  - GitHub Issue
  - Jira
  - ...
- Comparison to similar applications
  - DNF to YUM, ZYPPER

Impact

- Significant improvement of particular use-case
  - # dnf upgrade --bugfix
- Improvement in often used code
  - Queries for package name

Impact


45
40
35
30
25
20
15
10
5
0
10.03
10.31
2.38
40.95
40.28
2.41
dnf upgrade --security
dnf upgrade --bugfix
Dnf-1.1
Dnf-2.7.5
Dnf-4.0

Impact

- Significant improvement of particular use-case
  - # dnf upgrade --bugfix
- Improvement in often used code
  - Queries for package name

Performance measurement

```
class Timer(object):
  def __init__(self, msg=""):
    self.msg = msg
    self.start = 0

  def __enter__(self):
    self.start = time.time()
    return self

  def __exit__(self, *args):
    end = time.time()
    secs = end - self.start
    msecs = secs * 1000  # millisecs
    print('{}: Elapsed time: {} ms'.format(self.msg, msecs))
```

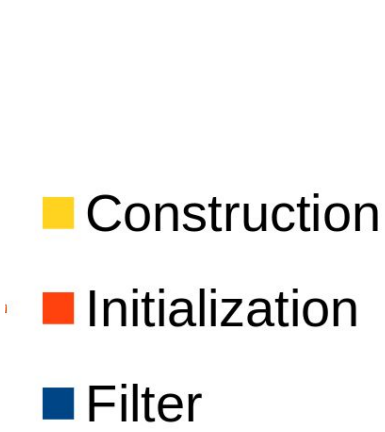
Usage in code:

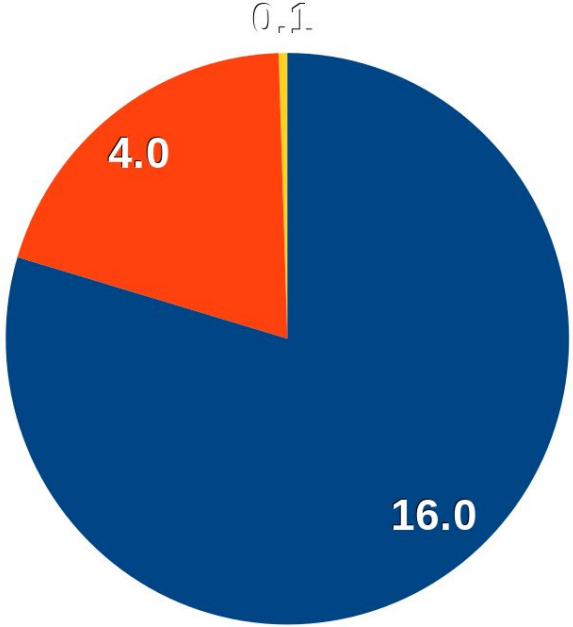
```
with Timer(msg="fill_sack"):
        base.fill_sack(load_system_repo=True)
```

Performance improvements - workflow

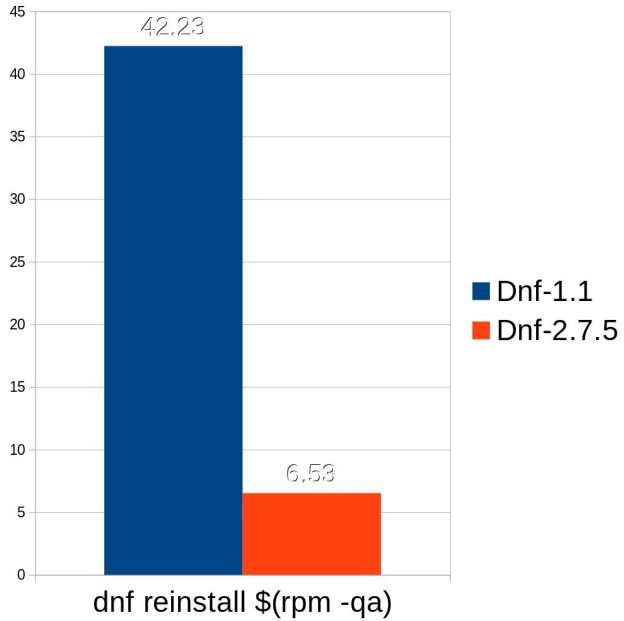
Performance analyses - name queries

```
#  constructor
query = base.sack.query()
#  query initialization
query.apply()
#  name search
query.filterm(name=[“dnf”]).apply()
```

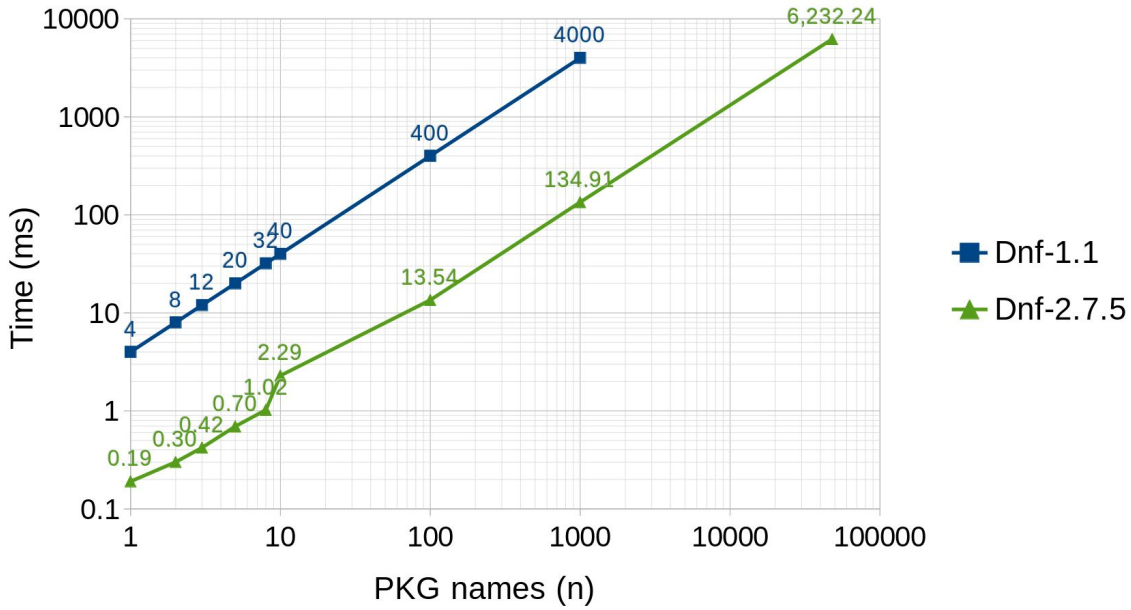




Performance analyses - name queries



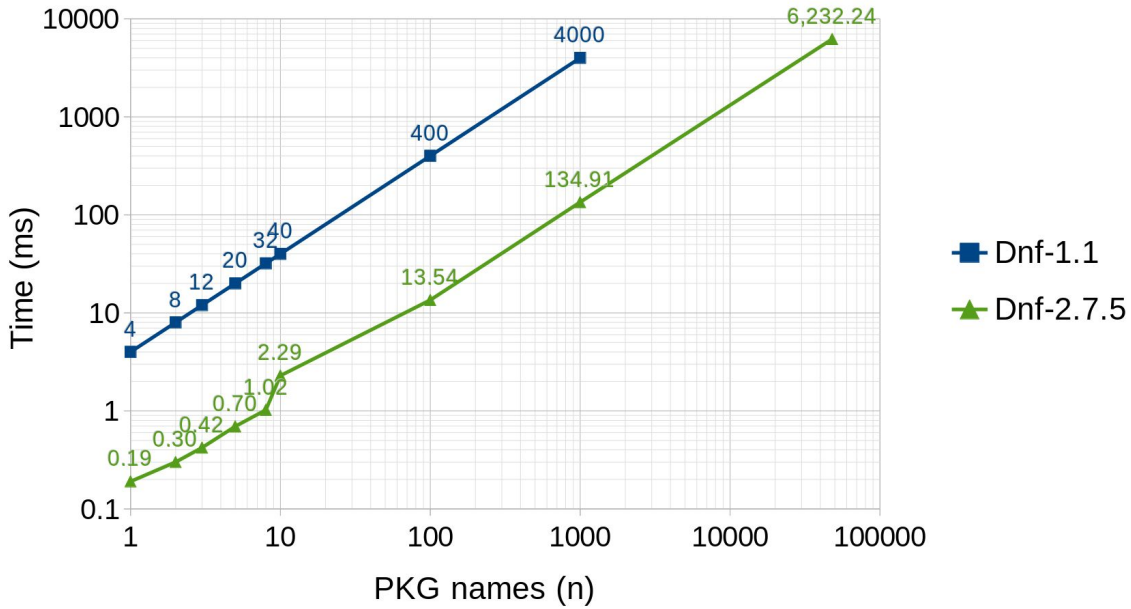
Performance analyses - name queries



Name queries - New requirements

- Additional queries for each run of DNF
- Arguments > 10 000
- No significant impact on DNF performance

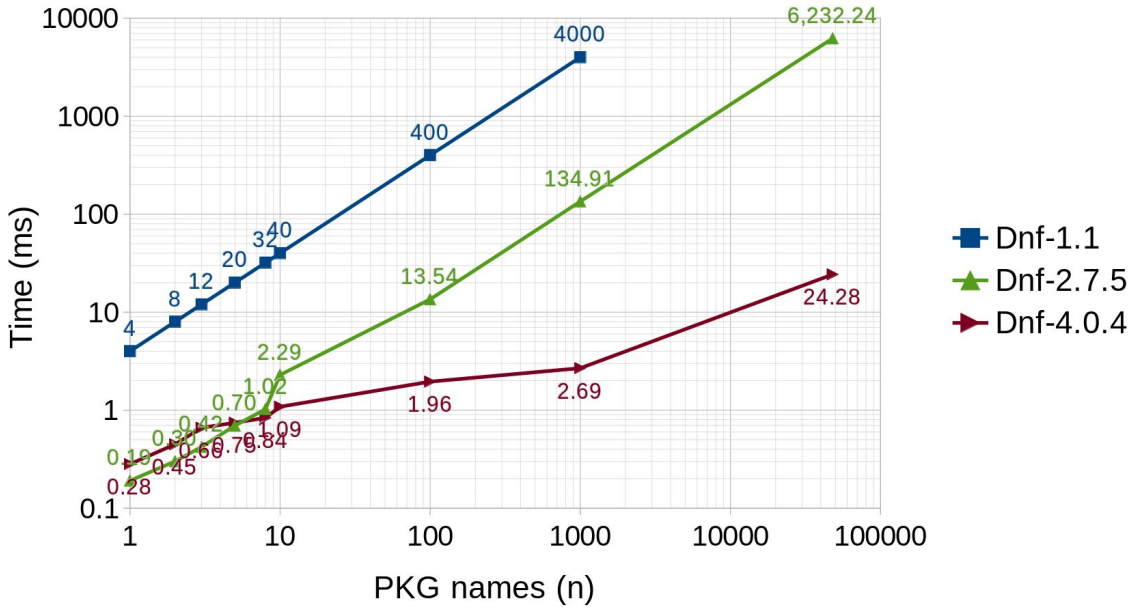
Performance analyses - name queries



Binary search

Search x = 91	1	5	15	34	48	69	74	88	91	98
Test 48 < x (True)	1	5	15	34	48	69	74	88	91	98
Test 88 < x (True)	1	5	15	34	48	69	74	88	91	98
Test 91 < x (False)	1	5	15	34	48	69	74	88	91	98
Test 91 == x (True)										

Performance analyses - name queries



Best Practise vs. Performance

The same process, different, requirements

=> different implementation

Example: String formatting
(“dnf-0:4.0.1-1.f31.noarch”)

name = “dnf”

epoch = “0”

version = “4.0.1”

release = “1.f31”

arch = “noarch”

Python example:

pattern = “{{}-{}:{}-{}.{}}”

pattern.format(name, epoch, version, release, arch)

Summary

- Work efficiently
- Prioritise users needs
- Look for the best potential
- Analyze
- Trade resources wisely
- Present results