

# Python pvAccess Performance Tests

Thomas Fors <tfors@aps.anl.gov>

February 7, 2018

## 1 Overview

Two competing packages exist to provide an interface to the pvAccess library of EPICS v4 from the Python language:

- [p4p](#)
- [pvaPy](#)

In this report, we compare their performance at handling monitors in three test situations.

All test code and scripts used to build or install dependencies for these tests are included in the git repository at <https://git.aps.anl.gov/tfors/python-pva-test>. Further details about the test environment and the scripts used may be found in section 5 at the end of this report.

## 2 Performance Tests

Three tests are run:

```
Spam Test 20 of 20 (100%) |#####| Elapsed Time: 0:21:44 Time: 0:21:44
Scale Test 40 of 40 (100%) |#####| Elapsed Time: 0:44:24 Time: 0:44:24
Array Test 64 of 64 (100%) |#####| Elapsed Time: 1:10:09 Time: 1:10:09
```

The tests use each Python package to create one or more subscriptions to the PVs for the given test and wait for all callbacks to fire at least once. Statistics are then gathered for a one-minute interval and the number of received and missed counts are written to a file.

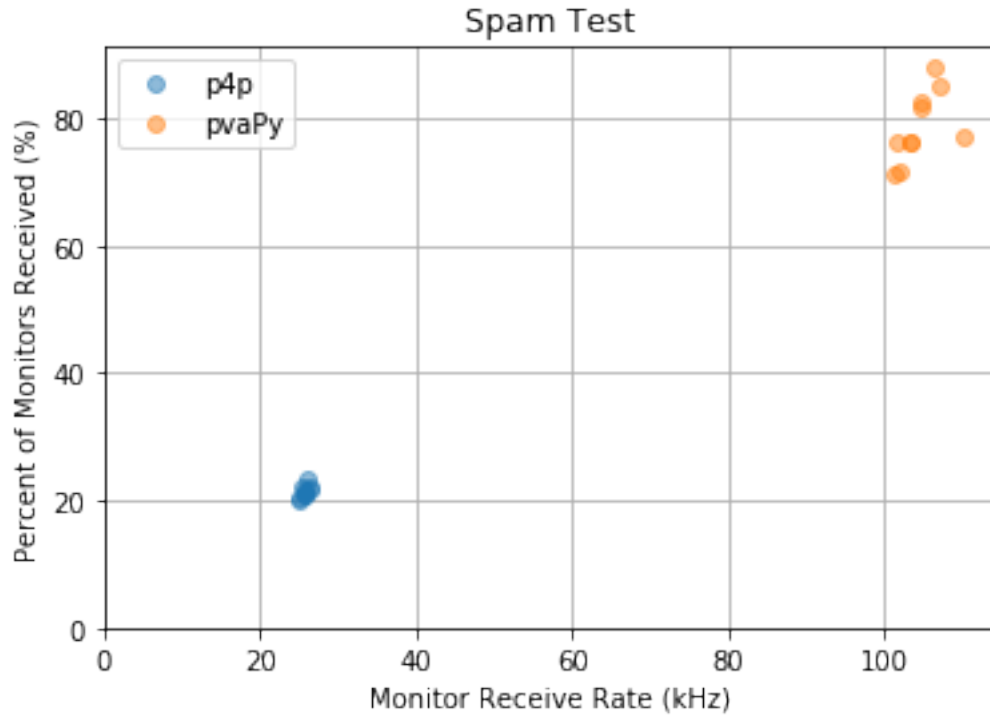
These tests were inspired by the examples folder of the pvaPy GitHub repository.

### 2.1 Spam Test

The spam test consists of running a soft IOC with a single record that increments a counter as fast as it can:

```
record(calc, "Spam") {
    field(INPA, "Spam.VAL CP")
    field(CALC, "A+1")
}
```

The clients, pvaPy and p4p were each run ten times and the results are shown below as a scatter plot of the percentage of monitors received vs. the average monitor receive rate over each one-minute test iteration.



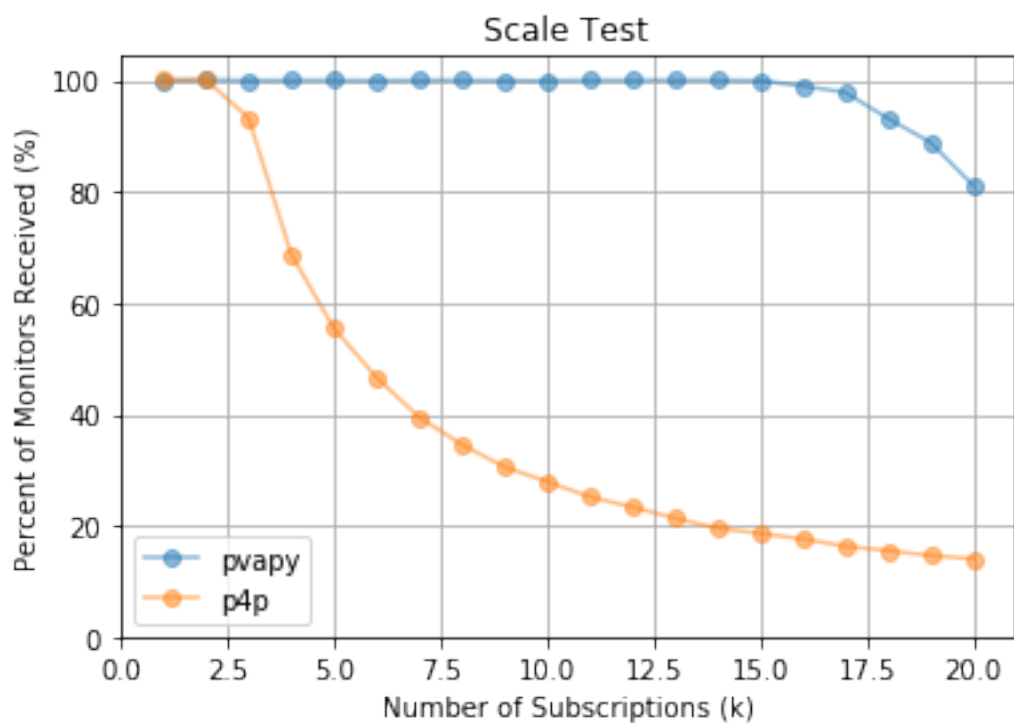
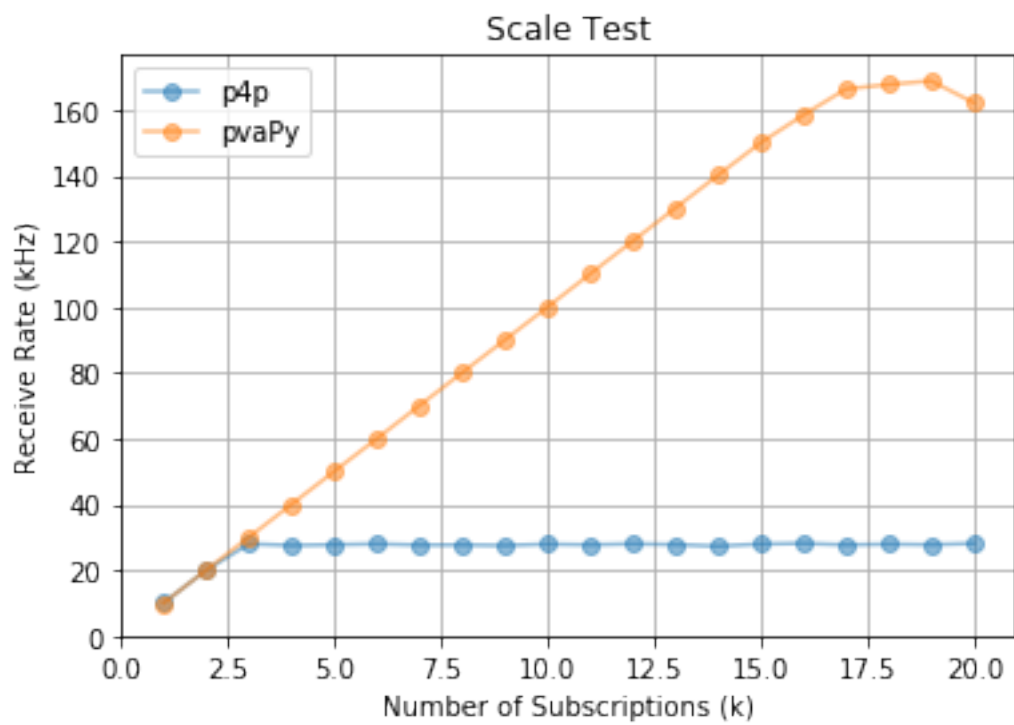
## 2.2 Scale Test

The scale test consists of running a soft IOC with a single record that increments a counter at a 10Hz rate:

```
record(calc, "Scale$(N)") {
    field(INPA, "Scale$(N).VAL")
    field(CALC, "A+1")
    field(SCAN, ".1 second")
}
```

Clients create  $N$  subscriptions to this record, wait for each monitor to fire at least once, and then record the received and missed counts for a one-minute interval. The test is repeated for  $N$  varying from 1000 to 20000.

The plots below show the aggregate monitor receive rate and the percent of monitors received as  $N$ , the number of subscriptions, varies.



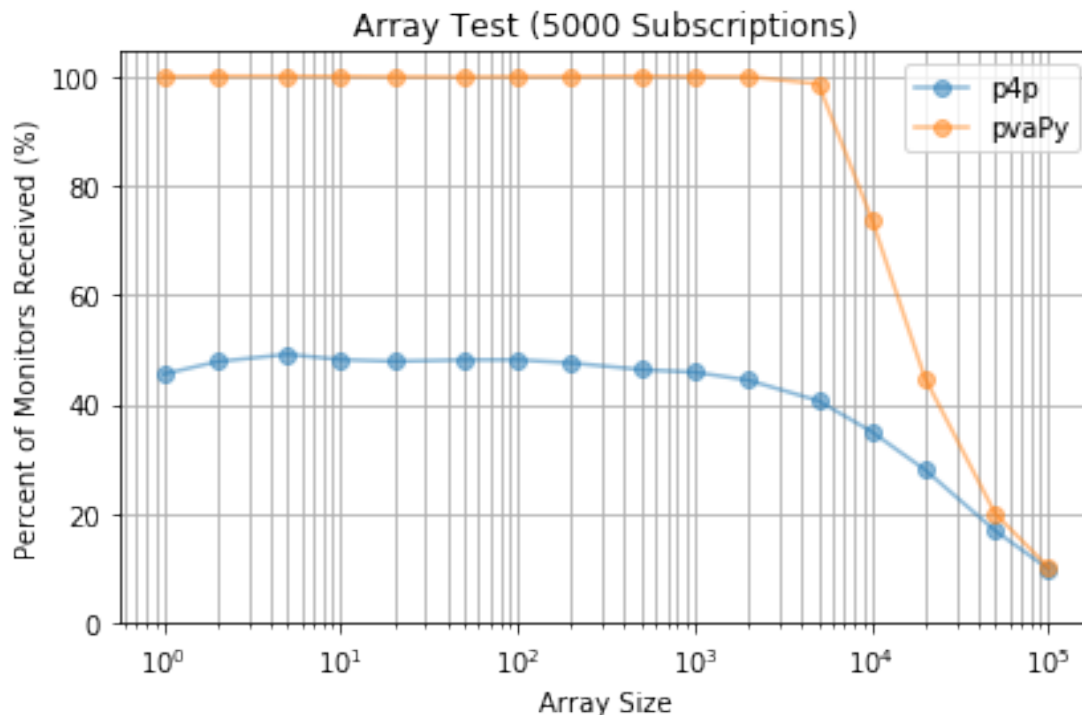
## 2.3 Array Test

The array test consists of a soft IOC with a single array subroutine record which processes at 10Hz. When the record processes, it sets NEVA to the value of the A field, thus changing the length of it's output field VALA. Additionally, it increments a counter at the first element.

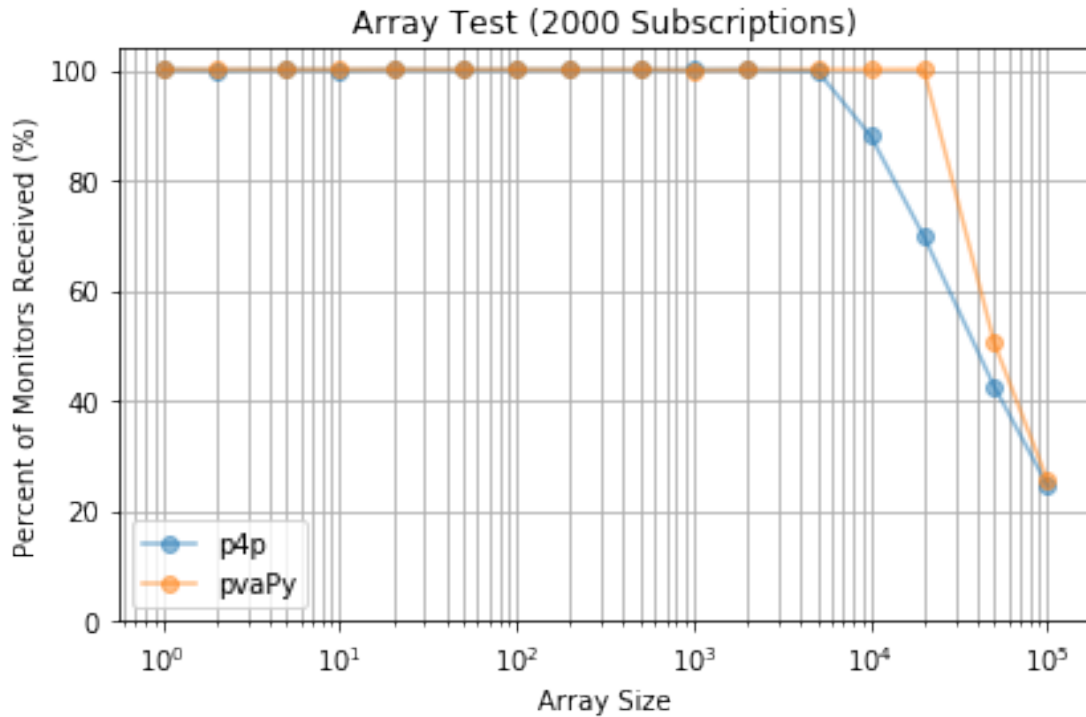
```
record(aSub, "Array") {  
    field(SNAM, "simDataSub")  
    field(FTA, "LONG")  
    field(NOA, "1")  
    field(INPA, "10")  
    field(FTVA, "DOUBLE")  
    field(NOVA, "1000000")  
    field(SCAN, ".1 second")  
}  
  
static long simDataSub(aSubRecord* prec) {  
    long n = ((long *) (prec->a))[0];  
    prec->neva = n;  
    double *p = prec->vala;  
    p[0] += 1;  
    return 0;  
}
```

Clients set the array size to  $N$ , create a given number of subscriptions to this record, wait for each monitor to fire at least once, and then record received and missed counts for a one-minute interval. The test is repeated for the array size,  $N$ , varying from 1 to 100000.

The plot below shows the percentage of monitors received as the array size,  $N$ , is varied using 5000 subscriptions in the clients. p4p is seen to drop monitors even for an array size of one which agrees with the results for the scale test.



The test was repeated using 2000 subscriptions in order to measure the client performance as a result of array size more directly.



### 3 Comments

In practice, usage of pvaPy and p4p for creating monitors is similar enough that a single client script was used for each test with a flag to determine which one to use.

PVA server and RPC performance was not tested, nor was performance using channel access as the provider.

p4p was consistently able to crash the soft IOC with a Seg-Fault during the scale test. It was determined that this happens only after the test has completed and the library is attempting to disconnect from the large number of subscriptions. Numerous attempts were made to work-around this problem before the client script exits such as calling `close()` on all the subscriptions and on the context, however, the most reliable solution found was to simply restart the IOC between test iterations. A minimal reproducible example will be created and a bug report will be filed on this issue.

After the initial results were obtained, the `maxsize` parameter to `p4p.client.thread.Context()` was discovered and it was adjusted in an attempt to improve the performance of p4p on the scale test, but this was unsuccessful.

## 4 Conclusion

The performance of pvAccess monitors using both p4p and pvaPy Python modules was tested in three test situations: spam, scale, and array tests. The results shown here indicate that pvaPy outperforms p4p in its capacity to process monitors in all three of these tests.

## 5 Test Environment

### 5.1 Preparing the test environment

Build scripts and test code to duplicate these tests are located in the git repository at <https://git.aps.anl.gov/tfors/python-pva-test>. In general, the steps to build the code and prepare the environment for running the tests are:

```
git clone https://git.aps.anl.gov/tfors/python-pva-test.git
cd python-pva-test
make
source setup.sh
make -C ioc
```

You should now be ready to run the tests.

#### 5.1.1 Python

Python 3.6.3 from the Anaconda distribution was used for these tests. The script `scripts/install_miniconda.sh` downloads and installs it.

#### 5.1.2 EPICS Base

EPICS Base version 7.0.1.1 was used for these tests. The script `scripts/build_epics_base.sh` downloads and builds it.

#### 5.1.3 p4p

p4p was fetched from the Git repository <https://github.com/mdavidsaver/p4p.git> using commit hash `aca1ef9` dated Dec. 20, 2017. The script `scripts/build_p4p.sh` downloads and builds it.

#### 5.1.4 pvaPy

pvaPy has an additional dependency: Boost.Python. The script `scripts/build_boost.sh` downloads and builds it.

pvaPy was fetched from the Git repository <https://github.com/epics-base/pvaPy.git> using commit hash `8854bb3` dated Nov. 17, 2017. The script `scripts/build_pvaPy.sh` downloads and builds it.

### 5.1.5 Jupyter Notebook (optional)

To load and execute this notebook file (Python `pvAccess Performance Tests.ipynb`), Jupyter Notebook and Matplotlib must be installed first. The script `scripts/install_jupyter.sh` downloads and installs it.

Then, running the command `jupyter notebook` can be used.

## 5.2 Test Scripts

The following test client scripts are provided:

- `scripts/spamClient.py`
- `scripts/scaleClient.py`
- `scripts/arrayClient.py`

Each runs a single test using either the `p4p` or `pvaPy` client as specified by a command-line flag and optionally appends its results to a datafile also specified on the command-line. Run these scripts with the `-h` parameter for a list of options.

IOCs for each test are also provided:

- `ioc/iocBoot/siocSpam/st.cmd`
- `ioc/iocBoot/siocScale/st.cmd`
- `ioc/iocBoot/siocArray/st.cmd`

Finally, scripts are provided to run multiple tests using each client, restart the IOCs between each test, and vary key parameters across each test.

- `scripts/runSpamTests.py`
- `scripts/runScaleTests.py`
- `scripts/runArrayTests.py`

Again, run these scripts with the `-h` parameter for a list of options.

## 5.3 Machine Details

The test machine used to obtain these results consists of the following CPU, memory, and operating system:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	4
On-line CPU(s) list:	0-3
Thread(s) per core:	2
Core(s) per socket:	2
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	142
Model name:	Intel(R) Core(TM) i7-7560U CPU @ 2.40GHz
Stepping:	9
CPU MHz:	2197.875
CPU max MHz:	3800.0000
CPU min MHz:	400.0000

BogoMIPS: 4799.76  
Virtualization: VT-x  
L1d cache: 32K  
L1i cache: 32K  
L2 cache: 256K  
L3 cache: 4096K  
NUMA node0 CPU(s): 0-3  
MemTotal: 8057696 kB  
No LSB modules are available.  
Distributor ID: Ubuntu  
Description: Ubuntu 16.04.3 LTS  
Release: 16.04  
Codename: xenial