MmStats Documentation

Release 0.8.0

Michael Schurter

November 20, 2013

Contents

Documentation | Package | Code **Not under active development**

CHAPTER 1

About

Mmstats is a way to expose and read diagnostic values and metrics for applications.

Think of mmstats as /proc for your application and the readers as procps utilities.

This project is a Python implementation, but compatible implementations can be made in any language (see Goals).

Discuss at https://groups.google.com/group/python-introspection

1.1 Goals

- Separate publishing/writing from consuming/reading tools
- Platform/language independent (a Java writer can be read by a Python tool)
- Predictable performance impact for writers via:
  - No locks (1 writer per thread)
  - No syscalls (after instantiation)
  - All in userspace
  - Reading has no impact on writers
- Optional persistent (writer can sync anytime)
- 1-way (Publish/consume only; mmstats are not management extensions)

CHAPTER 2

Usage

2.1 Requirements

CPython 2.6 or 2.7 (Windows is untested)

PyPy (only tested in 1.7, should be faster in 1.8)

2.2 Using

1. `easy_install mmstats` or `pip install mmstats` or if you've downloaded the source: `python setup.py install`
2. Then in your Python project create a sublcass of mmstats.MmStats like

```
import mmstats

class WebStats(mmstats.MmStats):
    status2xx = mmstats.CounterField(label='status.2XX')
    status3xx = mmstats.CounterField(label='status.3XX')
    status4xx = mmstats.CounterField(label='status.4XX')
    status5xx = mmstats.CounterField(label='status.5XX')
    last_hit = mmstats.DoubleField(label='timers.last_hit')
```

3. Instantiate it once per process: (instances are automatically thread local)

```
webstats = WebStats(label_prefix='web.stats.')
```

4. Record some data:

```
if response.status_code == 200:
    webstats.status2xx.inc()

webstats.last_hit = time.time()
```

5. Run `slurpstats` to read it
6. Run `mmash` to create a web interface for stats
7. Run `pollstats -p web.stats.status 2XX,3XX,4XX,5XX /tmp/mmstats-*` for a vmstat/dstat like view.

8. Did a process die unexpectedly and leave around a stale mmstat file? `cleanstats /path/to/mmstat/files` will check to see which files are stale and remove them.

CHAPTER 3

History

3.1 0.8.0

- `MmStats.remove` now removes all files for a PID if you use the TID in the filenames (thanks @bitprophet!)
- *mmash* now prepends `MMSTATS_PATH` to `?glob=...` parameters so users can't traverse the filesystem.
- Javascript fixes for *mmash*'s graphs (thanks @spiccinini!)
- Handle broken pipes in *slurpstats*

3.2 0.7.2 "Mr. Clean" released 2012-12-12

- *cleanstats* now cleans mmstats files for alive PIDs that are owned by other users.
- Minor cleanups to tests

3.3 0.7.1 "Mash Tun" released 2012-11-19

- `cleanstats` now defaults to `DEFAULT_GLOB` if no files are passed on the command line
- `CounterField.inc(n=...)` has been deprecated in favor of `CounterField.incr(amount=...)`
- *mmash* Improvements
  - Added `nonzero-avg` and `nonzero-min` aggregators to mmash to filter 0s out of aggregated metrics.
  - Added a `glob` query parameter to `/stats/<stats>` to filter which mmstats files are included in the stats
  - Backward incompatible change: switched `/stats/` to return a JSON Object instead of an array. The array is now the value of the `stats` key.
- Minor documentation and code cleanups

3.4 0.7.0 “Local Artisanal Stats” released 2012-10-02

- Per-thread model instances are created automatically - no need to manually create one per thread
- Backward incompatible change to naming templates; they now use str.format style substitutions:
  - New: `{CMD}` - Current process name
  - `%PID%` -> `{PID}`
  - `%TID%` -> `{TID}`

3.5 0.6.2 “Graphtastic” released 2012-03-23

- Added live graphing of numeric metrics thanks to @haard’s work at PyCon
- Documentation improvements

3.6 0.6.1 “MANIFEST.out” released 2012-03-08

- Fix packaging issue

3.7 0.6.0 “PyCon 2012” released 2012-03-08

- [API CHANGE] - MovingAverageField’s kwarg changed from window_size => size
- Refactored __init__.py into fields, models, and default (and imported public bits into __init__)
- Added TimerField (MovingAverageField + context manager)
- Added docs (don’t get too excited, just a start)

CHAPTER 4

Further topics

4.1 API

4.1.1 Models

class mmstats.models.**BaseMmStats**(*path='/tmp'*, *filename='{CMD}-{PID}-{TID}.mmstats'*, *label_prefix=None*)

Stats models should inherit from this

Optionally given a filename or label_prefix, create an MmStats instance

Both *filename* and *path* support the following variable substiutions:

- *{CMD}* - name of application (*os.path.basename(sys.argv[0])*)
- *{PID}* - process's PID (*os.getpid()*)
- *{TID}* - thread ID (tries to get it via the *SYS_gettid* syscall but fallsback to the Python/pthread ID or 0 for truly broken platforms)

This class is *not threadsafe*, so you should include both {PID} and {TID} in your filename to ensure the mmaped files don't collide.

flush(*async=False*)

Flush mmapped file to disk

Parameters **async** (*bool*) – `True` means the call won't wait for the flush to finish syncing to disk. Defaults to `False`

class mmstats.models.**FieldState**(*field*)

Holds field state for each Field instance

class mmstats.models.**MmStats**(*path='/tmp'*, *filename='{CMD}-{PID}-{TID}.mmstats'*, *label_prefix=None*)

Mmstats default model base class

Just subclass, add your own fields, and instantiate:

```
>>> from mmstats.models import MmStats
>>> from mmstats.fields import CounterField
>>> class MyStats(MmStats):
...     errors = CounterField()
```

```
...
>>> stats = MyStats()
>>> stats.errors.incr()
>>> stats.errors.value
1L
```

4.1.2 Model Fields

mmstats.fields.**AverageField**
Average field supporting an add() method and value attribute

mmstats.fields.**BoolField**
Boolean Field

mmstats.fields.**BufferedDescriptorField**
Base class for double buffered descriptor fields

mmstats.fields.**BufferedDescriptorMixin**
Mixin to add double buffered descriptor methods

Always read/write as double buffering doesn't make sense for readonly fields

mmstats.fields.**ByteField**
8bit Signed Integer Field

mmstats.fields.**ComplexDoubleBufferedField**
Base Class for fields with complex internal state like Counters

Set InternalClass in your subclass

mmstats.fields.**CounterField**
Counter field supporting an inc() method and value attribute

mmstats.fields.**DataDescriptorMixin**
Mixin to add single buffered __set__ method

class mmstats.fields.**DoubleBufferedField**(*label=None*)
Base class for double buffered writable fields

mmstats.fields.**DoubleField**
64bit Double Precision Float Field

exception mmstats.fields.**DuplicateFieldName**
Cannot add 2 fields with the same name to MmStat instances

mmstats.fields.**FloatField**
32bit Float Field

mmstats.fields.**IntField**
32bit Double Buffered Signed Integer field

mmstats.fields.**NonDataDescriptorMixin**
Mixin to add single buffered __get__ method

mmstats.fields.**ReadWriteField**
Base class for simple writable fields

mmstats.fields.**ShortField**
16bit Double Buffered Signed Integer field

mmstats.fields.**StaticDoubleField**
Unbuffered read-only 64bit Float field

mmstats.fields.**StaticFloatField**
Unbuffered read-only 32bit Float field

mmstats.fields.**StaticInt64Field**
Unbuffered read-only 64bit Signed Integer field

mmstats.fields.**StaticTextField**
Unbuffered read-only UTF-8 encoded String field

mmstats.fields.**StaticUInt64Field**
Unbuffered read-only 64bit Unsigned Integer field

mmstats.fields.**StaticUIntField**
Unbuffered read-only 32bit Unsigned Integer field

mmstats.fields.**StringField**
UTF-8 String Field

mmstats.fields.**TimerField**
Moving average field that provides a context manager for easy timings

As a context manager:

```
>>> class T(MmStats):
...     timer = TimerField()
>>> t = T()
>>> with t.timer as ctx:
...     assert ctx.elapsed > 0.0
>>> assert t.timer.value > 0.0
>>> assert t.timer.last > 0.0
```

mmstats.fields.**UInt64Field**
Unbuffered read-only 64bit Unsigned Integer field

mmstats.fields.**UIntField**
32bit Double Buffered Unsigned Integer field

mmstats.fields.**UShortField**
16bit Double Buffered Unsigned Integer field

4.1.3 Reader API

mmstats reader implementation

exception mmstats.reader.**InvalidMmStatsVersion**
Unsupported mmstats version

class mmstats.reader.**Stat**
Stat(label, value)

label
Alias for field number 0

value
Alias for field number 1

4.1.4 Internal Defaults

4.1.5 mmap Compatibility Wrapper

class mmstats._mmap.**MmapInfo**
MmapInfo(fd, size, pointer)

fd
Alias for field number 0

pointer
Alias for field number 2

size
Alias for field number 1

mmstats._mmap.**init_mmap**(*filename*, *size=4096*)
Create an mmap given a location *filename* and minimum *size* in bytes

Returns an MmapInfo tuple with the file descriptor, actual size, and a pointer to the begging of the mmap.

Note that the size returned is rounded up to the nearest PAGESIZE.

4.2 Development

4.2.1 Getting Started

It's easiest to develop mmstats within a virtualenv:

```
$ git clone git://github.com/schmichael/mmstats.git
$ cd mmstats
$ virtualenv .
$ source bin/activate
$ python setup.py develop
$ ./run_flask_example # This starts up a sample web app
$ curl http://localhost:5001/
$ curl http://localhost:5001/500
$ curl http://localhost:5001/status
$ # If you have ab installed:
$ ab -n 50 -c 10 http://localhost:5001/
```

Now to view the stats run the following in a new terminal:

```
$ # To get a raw view of the data:
$ slurpstats mmstats-*
$ # Or start up the web interface:
$ mmash
$ # Run pollstats while ab is running:
$ pollstats -p flask.example. ok,bad,working mmstats-*
```

To cleanup stray mmstats files: `rm mmstats-flask-*`

The web interface will automatically reload when you change source files.

Put static files into static/ and template files into templates/

Testing

Feel free to use your favorite test runner like nose or pytest or just run:

```
$ python setup.py test
```

4.2.2 TODO

There's always bugs to fix: https://github.com/schmichael/mmstats/issues/

- Add API to dynamically add fields to MmStat classes
- Percentiles
- Time based windows for moving averages (eg last 60 seconds)
- Multiple exposed fields (average, mean, and percentiles) from 1 model field
- Add alternative procedural writer API (vs existing declarative models)
- Test severity of race conditions (especially: byte value indicating write buffer)
- Test performance
- Vary filename based on class name
- Improve mmash (better live graphing, read from multiple paths, etc)
- Include semantic metadata with field types (eg to differentiate an int that's a datetime from an int that's a counter)
- Logo

4.2.3 mmap Format

Structure of version 1 mmstat's mmaps:

version number	fields...
`byte = 01`	...

Fields

There are two types of field structures so far in mmstats:

1. buffered
2. unbuffered

Buffered fields use multiple buffers for handling values which cannot be written atomically.

Unbuffered structures have `ff` in the write buffer field.

Buffered

label size	label	type size	type	write buffer	buffer 1	buffer 2
`ushort`	`char[]`	`ushort`	`char[]`	`byte`	varies	varies

The buffers field length = sizeof(type) * buffers.

The current write buffer is referenced by: write_buffer * sizeof(type)

TODO: field for total number of buffers?

Unbuffered

label size	label	type size	type	write buffer	value
ushort	char[]	ushort	char[]	byte = ff	varies

The value field length = sizeof(type).

4.2.4 Scrapped Ideas

Compounds Fields where 1 Writer Field = Many Mmap/Reader Fields

This seemed like a honking great idea at first. Compound fields would look just like a mini-MmStat model:

```
class SamplingCounterField(CompoundField):
    """Records increments per ms every N increments"""
    counter = CounterField()
    per_ms = UInt64Field()

    class _Counter(object):
        """Implement counter/rate-sampling logic here"""

    def __get__(self, inst, owner):
        if inst is None:
            return self
        return inst._fields[self.key]._counter_instance
```

The blocker is that there's no way to atomically update all of the compound fields. The only way to accomplish this is for compound fields to appear as a single double buffered field with each component field as a type in the type signature:

```
class SamplingCounterField(DoubleBufferedField):
    initial = (
        CounterField.initial,
        UInt64Field.initial,
    )
    buffer_type = (
        CounterField.buffer_type,
        UInt64Field.buffer_type,
    )
    type_signature = (
        CounterField.type_signature + UInt64Field.type_signature
    )
```

Obviously an actual implementation should remove the redundant references to the component types.

Note: Lack of atomicity is not a blocker for exposing fields such as Mean, Median, and Percentiles.

Solution: Future versions of the mmstats format should support structs as values instead of just scalars so that a single write buffer offset can point to multiple values.

Metadata metaprogramming

To get around having to dynamically creating the structs due to a variable label size, put the labels in a header index with a pointer to the actual struct field.

Page Flipping

Store metadata seperate from values. Then store values in multiple pages and flip between pages for read/write buffering.

See *Getting Started* for how to get started hacking on mmstats or *TODO* for a list of ways you can help.

4.3 History

4.3.1 0.8.0

- `MmStats.remove` now removes all files for a PID if you use the TID in the filenames (thanks @bitprophet!)
- *mmash* now prepends `MMSTATS_PATH` to `?glob=...` parameters so users can't traverse the filesystem.
- Javascript fixes for *mmash*'s graphs (thanks @spiccinini!)
- Handle broken pipes in *slurpstats*

4.3.2 0.7.2 "Mr. Clean" released 2012-12-12

- *cleanstats* now cleans mmstats files for alive PIDs that are owned by other users.
- Minor cleanups to tests

4.3.3 0.7.1 "Mash Tun" released 2012-11-19

- `cleanstats` now defaults to `DEFAULT_GLOB` if no files are passed on the command line
- `CounterField.inc(n=...)` has been deprecated in favor of `CounterField.incr(amount=...)`
- *mmash* Improvements
  - Added `nonzero-avg` and `nonzero-min` aggregators to mmash to filter 0s out of aggregated metrics.
  - Added a `glob` query parameter to `/stats/<stats>` to filter which mmstats files are included in the stats
  - Backward incompatible change: switched `/stats/` to return a JSON Object instead of an array. The array is now the value of the `stats` key.
- Minor documentation and code cleanups

4.3.4 0.7.0 "Local Artisanal Stats" released 2012-10-02

- Per-thread model instances are created automatically - no need to manually create one per thread
- Backward incompatible change to naming templates; they now use str.format style substitutions:
  - New: `{CMD}` - Current process name
  - `%PID%` -> `{PID}`
  - `%TID%` -> `{TID}`

4.3.5 0.6.2 "Graphtastic" released 2012-03-23

- Added live graphing of numeric metrics thanks to @haard's work at PyCon
- Documentation improvements

4.3.6 0.6.1 "MANIFEST.out" released 2012-03-08

- Fix packaging issue

4.3.7 0.6.0 "PyCon 2012" released 2012-03-08

- [API CHANGE] - MovingAverageField's kwarg changed from window_size => size
- Refactored __init__.py into fields, models, and default (and imported public bits into __init__)
- Added TimerField (MovingAverageField + context manager)
- Added docs (don't get too excited, just a start)

4.3.8 0.5.0 "100% More Average" released 2012-02-25

- [API CHANGE] - RunningAverage field is now AverageField
- Added MovingAverageField with window_size=100 parameter
- Tests can now be run via "python setup.py test"

4.3.9 0.4.1 "Derpstats" released 2012-01-31

- Fixed pollstats
- Updated README slightly

4.3.10 0.4.0 "On the Road to Pycon" released 2012-01-17

- Added clean module and cleanstats script to clean stale mmstat files
- Added path kwarg to MmStats class to allow easy path overriding
- Added StringField for UTF-8 encoded strings
- Added StaticFloatField & StaticDoubleField
- Added created UNIX timestamp (sys.created) to default MmStats class
- Moved all modules into mmstats package
- Fixed mmash template packaging
- Fixed test mmstat file cleanup
- Refactored reading code into mmstats.reader module

4.3.11 0.3.12 "Meow" released 2011-11-29

- Use ctypes.get_errno() instead of Linux specific wrapper

4.3.12 0.3.11 "Rawr" released 2011-11-29

- Fix libc loading on OSX

4.3.13 0.3.10 "π^2" released 2011-11-28

- PyPy support (switched from ctypes._CData.from_buffer to .from_address)
- Multiple calls to MmStats().remove() no longer error (makes testing easier)

4.3.14 0.3.9 "MLIT" released 1970-01-01

- Mistag of 0.3.8

4.3.15 0.3.8 "Hapiness" released 2011-11-20

- Allow filename templating with %PID% and %TID% placeholders
- Allow setting filename template via MMSTATS_FILES environment variable
- Improved docs slightly
- Fixed Ctrl-Cing run_flask_example script
- Fixed 64 bit integer fields on 32 bit platforms
- Fixed StaticInt64Field (was actually a uint64 field before)
- Fixed slurpstats debug output (always showed first 40 bytes of file)
- Strip newlines from org.python.version

4.3.16 0.3.7 "Depressive Realism is for Winners" released 2011-11-17

- Add pollstats utility (similar to dstat/vmstat)
- Cleanup development/testing section of the README
- Slight improvements to basic flask integration example

4.3.17 0.3.6 "The M is for Mongo" released 2011-11-09

- Allow setting the value of CounterFields

4.3.18 0.3.5 "Ornery Orangutan" released 2011-10-20

- Added a running average field
- Made mmash more configurable and added a console entry point
- Updated TODO

4.4 Contributors

Copyright 2012 Urban Airship and contributors

4.4.1 Developers

- Dan Colish <dcolish@gmail.com>
- Jeff Forcier - https://github.com/bitprophet
- Fredrik Håård - https://github.com/haard
- Adam Lowry <adam@therobots.org>
- Santiago Piccinini - https://github.com/spiccinini
- Michael Schurter <m@schmichael.com> (creator/lead)

4.4.2 Reviewers

- Niall Kelly

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

m