Utilizing tools made for *Big Data* to analyse Ftrace data: making it fast and easy

Yordan Karadzhov

VMware Inc. - OSTC

Motivation

a. Ftrace: The official tracer of the Linux kernel (No need to explain this)

b. Is the Ftrace data *Big Data*?

  * Not necessarily. It depends how you use it.
  * Extremely sophisticated instrument. Large variety of use cases.

Motivation

a. OK, I have a nontrivial or very user-specific problem.

b. I have recorded a lot of tracing data.

c. What should I do now?

Motivation

a. OK, I have a nontrivial or very user-specific problem.

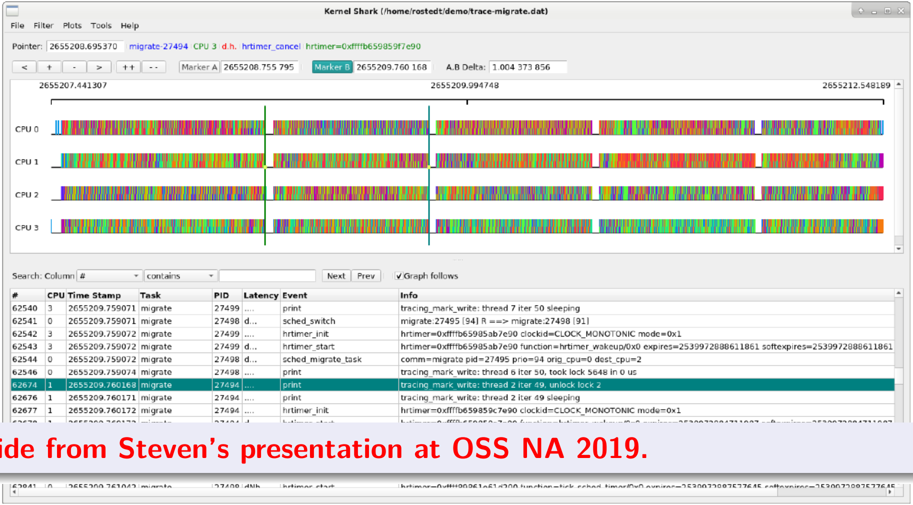
b. I have recorded a lot of tracing data.

c. What should I do now?

KernelShark

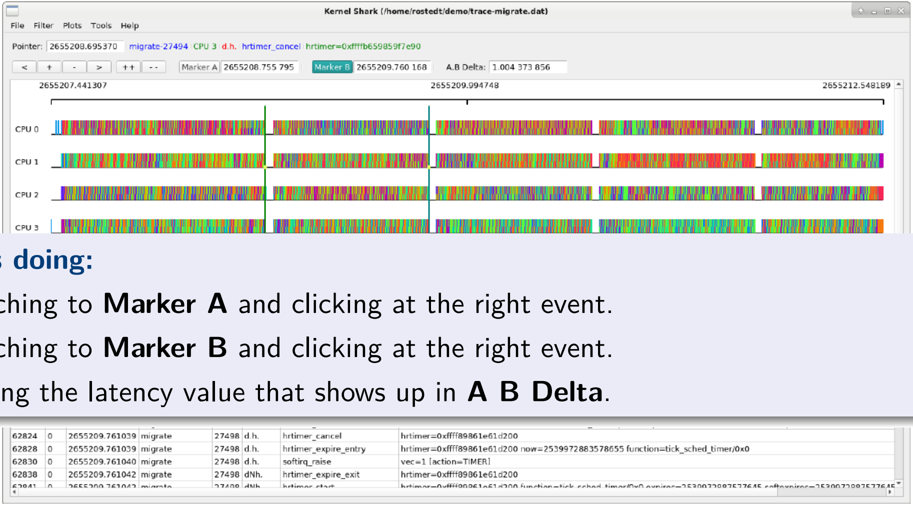
Not going to explain it here. See Steven Rostedt's presentation at OSS NA 2019.

KernelShark (Something More)



Stolen slide from Steven's presentation at OSS NA 2019.

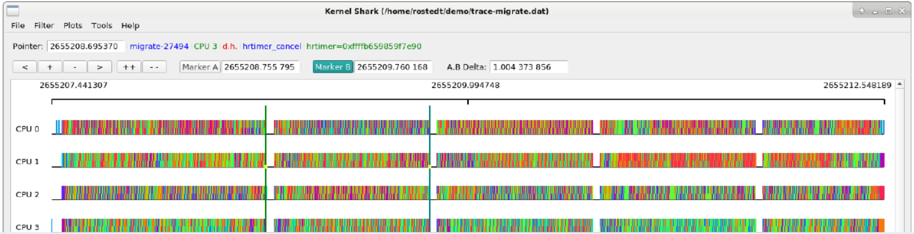
KernelShark (Something More)



Steven is doing:

* Switching to **Marker A** and clicking at the right event.
* Switching to **Marker B** and clicking at the right event.
* Getting the latency value that shows up in **A B Delta**.

KernelShark (Something More)



Steven is doing:

* Switching to **Marker A** and clicking at the right event.
* Switching to **Marker B** and clicking at the right event.
* Getting the latency value that shows up in **A B Delta**.

Now imagine doing this 10K times - Ugh!!!

There must be a better way to get this job done.

Imagine having something like this:

```python
#!/usr/bin/env python3
import ksharkpy as ks
...
ks.open_file('trace.dat')
data = ks.load_data()
data_size = ks.data_size(data)
...
for i in range(data_size):
    if data['event'][i] == my_event_a:
        action1
    elif data['event'][i] == my_event_b:
        action2
...
ks.close()
print('some summary of the results')
plot('some cool histograms or graphs for my presentation')
```

NumPy

a. **General purpose languages:** C, Perl, Python ...

b. **Numerical languages:** Fortran, MATLAB, R, ...
   * Written mostly for scientific numerical use.

Python + Scientific computing = NumPy

NumPy is not built in to the Python language. It is a library.

It provides:

1. Powerful densely packed N-dimensional arrays of homogeneous type.
2. Large collection of high-level mathematical functions to operate on these arrays.
3. Tools for integrating C/C++ and Fortran code.
4. Complementary packages like:
   a. Matplotlib - plotting package that provides MATLAB-like plotting functionality.
   b. SciPy - library that adds functionalities for optimization, linear algebra, integration, interpolation, FFT, signal and image processing.

NumPy

* Many Numpy operations are implemented in C.
* In fact the Numpy arrays are very similar to the C arrays.
* Numpy array can be initialized from C-computed array without data copying - COOL!!!

NumPy

* Many Numpy operations are implemented in C.
* In fact the Numpy arrays are very similar to the C arrays.
* Numpy array can be initialized from C-computed array without data copying - COOL!!!

Let's use something that is already (almost ;-) available in KernelShark

libkshark.so

Already available in KernelShark 1.0

Example of loading data using libkshark.so:

```
#include "libkshark.h"

int main(int argc, char **argv)
{
    struct kshark_context *kshark_ctx = NULL;
    struct kshark_entry **data = NULL;
    int data_size;

    kshark_instance(&kshark_ctx);
    kshark_open(kshark_ctx, "trace.dat");
    data_size = kshark_load_data_entries(kshark_ctx, &data);

    for (r = 0; r < data_size; ++r) {
        if (data[i]->event_id == my_event_a)
            action1;
        if (data[i]->event_id == my_event_b)
            action1;
    }
```

```c
...
    /* Free the memory. */
    for (r = 0; r < data_size; ++r)
        free(data[r]);
    free(data);

    /* Close the file. */
    kshark_close(kshark_ctx);

    /* Close the session. */
    kshark_free(kshark_ctx);

    printf("some summary of the results")
    /*
     * Unfortunately, no simple way to show cool histograms/graphs here :(
     */

    return 0;
}
```

Summary

a. PoC NumPy interface for accessing Ftrace data in Python (via NumPy arrays).

b. The implementation is just a tiny wrapper around **libkshark**

Summary

a. PoC NumPy interface for accessing Ftrace data in Python (via NumPy arrays).

b. The implementation is just a tiny wrapper around **libkshark**

Let's see some examples.

Example 1

Trying to reproduce the study done by
Prof. Dr. Wolfgang Mauerer and
Daniel Wagner.

See:

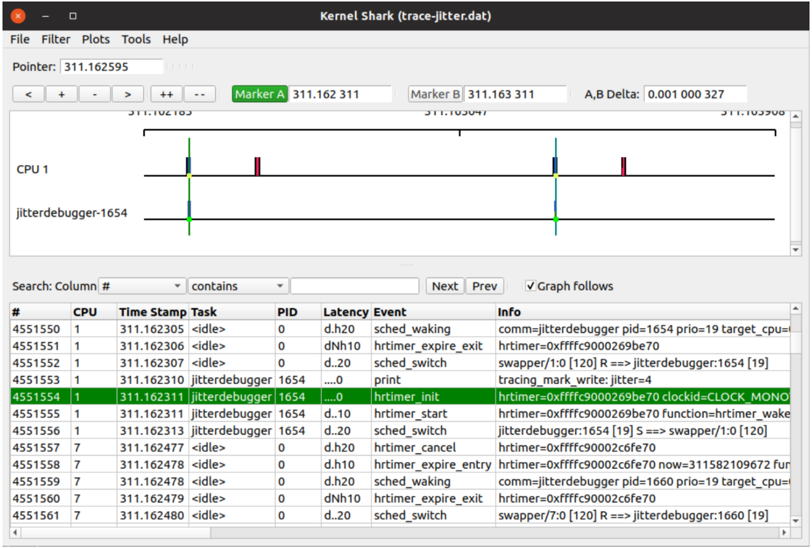
Cyclic Tests Unleashed: Large-Scale RT Analysis with Jitterdebugger
Open Source Summit Japan 2019

Example 1

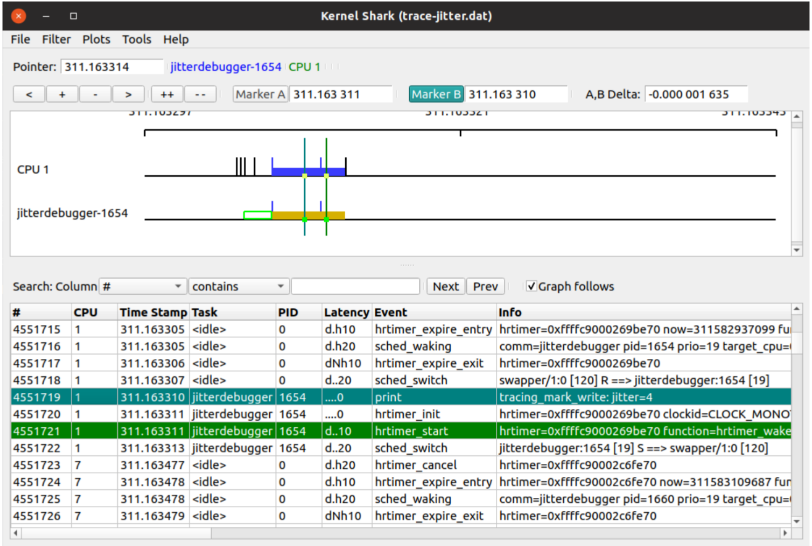
Goal:

* **Statistical estimate of the probability of exceeding the Worst Case Execution Time (WCET)**
* Remember that this is just an example demonstrating the PoC NumPy interface for Ftrace data. **The whole credit for the development of the analysis itself goes to Wolfgang and Daniel.**

Example 1: Jitterdebugger - Idle machine



Example 1: Jitterdebugger - Idle machine



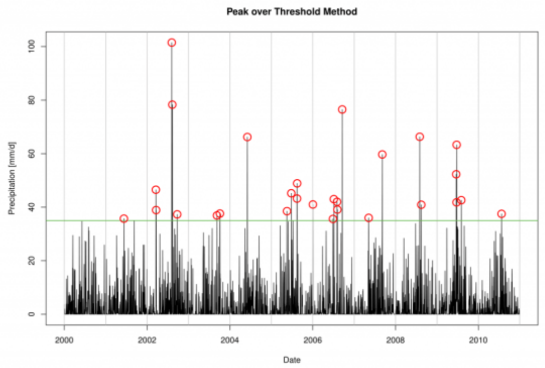
Example 1: Jitterdebugger

To make it more interesting, let's do the test on a heavy loaded system.

Hackbench: stress test for the Linux kernel scheduler.

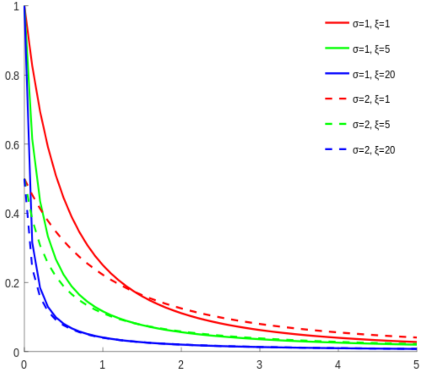
Example 1: Jitterdebugger

Extreme Value Theory: Peak over Threshold (PoT) approach



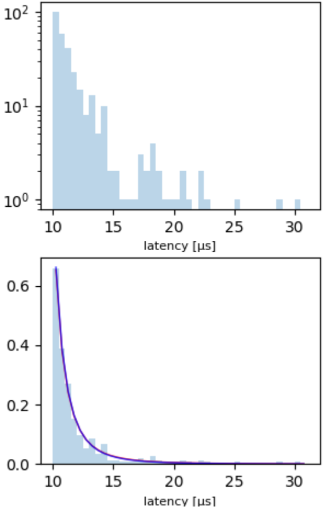
Example 1: Jitterdebugger

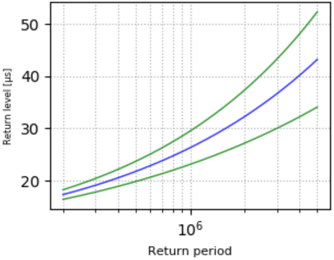
$$f(x) = \left(1 + \frac{\xi(x-\mu)}{\sigma}\right)^{-(1+1/\xi)}$$



Generalized Pareto distribution: A continuous probability distributions. It is often used to model the tails of other distributions.

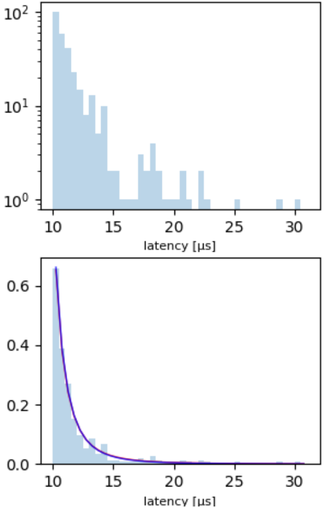
Example 1: Jitterdebugger ∼2.4M cycles

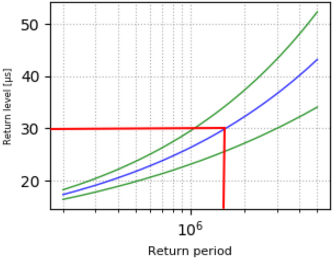




ξ = 0.393 +- 0.061 (15.53%)
σ = 1.124 +- 0.031 (2.77%)
µ = 10 (const)
goodness-of-fit: 1.797

Example 1: Jitterdebugger ~2.4M cycles

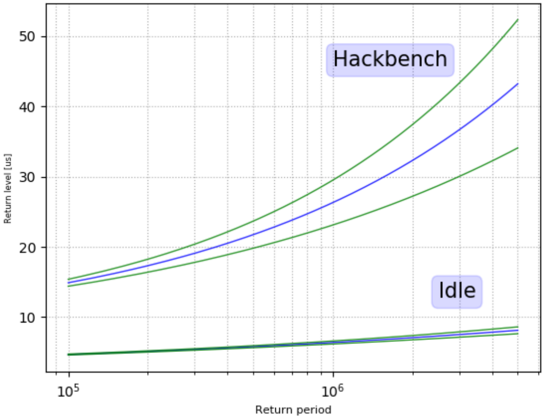




ξ = 0.393 +- 0.061 (15.53%)
σ = 1.124 +- 0.031 (2.77%)
μ = 10 (const)
goodness-of-fit: 1.797

Example 1: Jitterdebugger

Idle vs. Hackbench



Example 1: Jitterdebugger

Let's see some real code

KernelShark session description file (JSON)

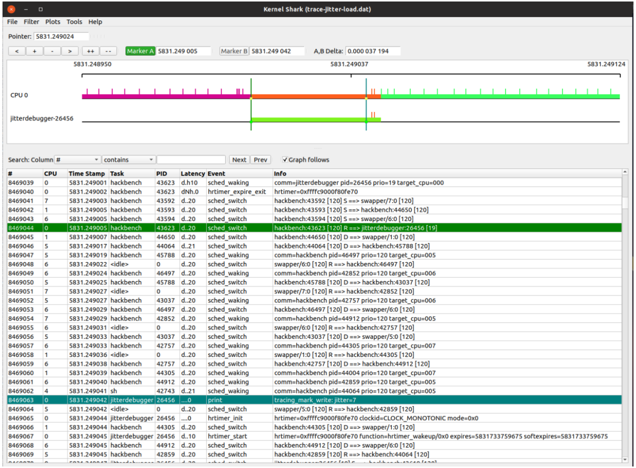
Allows loading predefined sessions.


```
{
    "type": "kshark.config.session",
    "Data": {
        "type": "kshark.config.data",
        "file": "trace-jitter-load.dat",
        "time": 1567097989
    },
    "Model": {
        "type": "kshark.config.model",
        "range": [
            5831246982148,
            5831250072340
        ],
        "bins": 1000
    },
    ...
```

```
    "Markers": {
        "type": "kshark.config.markers",
        "markA": {
            "isSet": true,
            "row": 8467891
        },
        "markB": {
            "isSet": true,
            "row": 8469063
        },
        "Active": "A"
    },
    "CPUPlots": [
        0
    ],
    "TaskPlots": [
        26456
    ],
    "ViewTop": 8467886
}
```

Example 1: Jitterdebugger

kernelshark -s max_lat.json



#	CPU	Time Stamp	Task	PID	Latency	Event	Info
8469039	0	5831.249001	hackbench	43623	d.h10	sched_waking	comm=jitterdebugger pid=26456 prio=19 target_cpu=000
8469040	0	5831.249002	hackbench	43623	dNh.0	hrtimer_expire_exit	hrtimer=0xffffc9000f80fe70
8469041	7	5831.249003	hackbench	43592	d..20	sched_switch	hackbench:43592 [120] S ==> swapper/7:0 [120]
8469042	1	5831.249005	hackbench	43593	d..20	sched_switch	hackbench:43593 [120] S ==> hackbench:44650 [120]
8469043	6	5831.249005	hackbench	43594	d..20	sched_switch	hackbench:43594 [120] S ==> swapper/6:0 [120]
8469044	0	5831.249005	hackbench	43623	d..20	sched_switch	hackbench:43623 [120] R ==> jitterdebugger:26456 [19]
8469045	1	5831.249007	hackbench	44650	d..20	sched_switch	hackbench:44650 [120] D ==> swapper/1:0 [120]
8469046	5	5831.249017	hackbench	44064	d..21	sched_switch	hackbench:44064 [120] D ==> hackbench:45788 [120]
8469047	5	5831.249019	hackbench	45788	d..20	sched_waking	comm=hackbench pid=46497 prio=120 target_cpu=005
8469048	6	5831.249022	<idle>	0	d..20	sched_switch	swapper/6:0 [120] R ==> hackbench:46497 [120]
8469049	6	5831.249024	hackbench	46497	d..20	sched_waking	comm=hackbench pid=42852 prio=120 target_cpu=006
8469050	5	5831.249025	hackbench	45788	d..20	sched_switch	hackbench:45788 [120] D ==> hackbench:43037 [120]
8469051	7	5831.249027	<idle>	0	d..20	sched_switch	swapper/7:0 [120] R ==> hackbench:42852 [120]
8469052	5	5831.249027	hackbench	43037	d..20	sched_waking	comm=hackbench pid=42757 prio=120 target_cpu=006
8469053	6	5831.249029	hackbench	46497	d..20	sched_switch	hackbench:46497 [120] D ==> swapper/6:0 [120]
8469054	7	5831.249029	hackbench	42852	d..20	sched_waking	comm=hackbench pid=44912 prio=120 target_cpu=005
8469055	6	5831.249031	<idle>	0	d..20	sched_switch	swapper/6:0 [120] R ==> hackbench:42757 [120]
8469056	5	5831.249033	hackbench	43037	d..20	sched_switch	hackbench:43037 [120] D ==> swapper/5:0 [120]
8469057	6	5831.249033	hackbench	42757	d..20	sched_waking	comm=hackbench pid=44305 prio=120 target_cpu=007
8469058	1	5831.249036	<idle>	0	d..20	sched_switch	swapper/1:0 [120] R ==> hackbench:44305 [120]
8469059	6	5831.249038	hackbench	42757	d..20	sched_switch	hackbench:42757 [120] D ==> hackbench:44912 [120]
8469060	1	5831.249039	hackbench	44305	d..20	sched_waking	comm=hackbench pid=44004 prio=120 target_cpu=007
8469061	6	5831.249040	hackbench	44912	d..20	sched_waking	comm=hackbench pid=42859 prio=120 target_cpu=005
8469062	4	5831.249041	sh	42743	d..21	sched_waking	comm=hackbench pid=44064 prio=120 target_cpu=005
8469063	0	5831.249042	jitterdebugger	26456	0	print	tracing_mark_write: jitter=7
8469064	5	5831.249042	<idle>	0	d..20	sched_switch	swapper/5:0 [120] R ==> hackbench:42859 [120]
8469065	0	5831.249044	jitterdebugger	26456	0	hrtimer_init	hrtimer=0xffffc9000f80fe70 clockid=CLOCK_MONOTONIC mode=0x0
8469066	1	5831.249044	hackbench	44305	d..20	sched_switch	hackbench:44305 [120] D ==> swapper/1:0 [120]
8469067	0	5831.249045	jitterdebugger	26456	d..10	hrtimer_start	hrtimer=0xffffc9000f80fe70 function=hrtimer_wakeup/0x0 expires=5831733759675 softexpires=5831733759675
8469068	6	5831.249045	hackbench	44912	d..20	sched_switch	hackbench:44912 [120] D ==> swapper/6:0 [120]
8469069	5	5831.249045	hackbench	42859	d..20	sched_switch	hackbench:42859 [120] R ==> hackbench:44064 [120]

Example 2: Page Faults

Demo