HITACHI
Inspire the Next

Analyzing Kernel Behavior by SystemTap

～ Kernel Tracer Approach ～

2009/2/25

Hitachi, Ltd., Software Division

Noboru Obata (小幡 昇)

uVALUE

HITACHI
Inspire the Next

Contents

1. Improving RAS Features for Linux®
2. Kernel Tracer Requirement
3. Kernel Tracer Prototype
4. Kernel Tracer for Production Use
5. Future Work

uVALUE

HITACHI
Inspire the Next

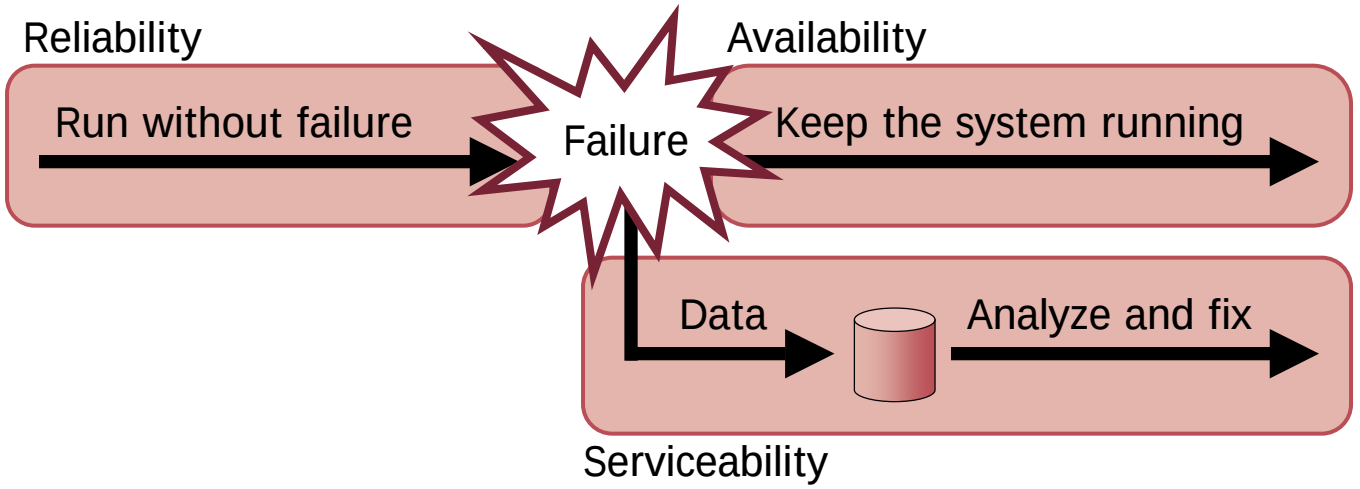
1

Improving RAS Features for Linux®

uVALUE

1-1. RAS Features

- Reliability: Components (hardware or software) do not break.
  - ECC memory, retry (__alloc_pages), reservation (mempoll), etc.
- Availability: Keep the system running even under the failure.
  - RAID, hardware hot-swap, machine check recovery, etc.
- Serviceablility: Diagnose the cause of failure correctly and quickly.
  - Collect enough information on failure.
  - /var/log/messages, kdump, Kernel Tracer, etc.



1-2. What's a Kernel Tracer?

HITACHI
Inspire the Next

- Kernel Tracer
  - Collect kernel events and associated data at all times in background, to diagnose the future problems quickly.
  - Events example: System calls, process switch, device I/O, etc.
- How Useful?
  - Panic analysis – Together with kernel memory dump (generated by kdump), the kernel trace data provides the kernel behavior just before the failure (for several seconds).

    It helps us narrowing down the cause of failure, and provides a fact to back up our assumption.
  - Performance analysis – Kernel trace data helps us understand what is going on inside the kernel when the system performance is not satisfactory.

    It helps us pinpoint the problem, rather than repeating the performance test over and over again.

1-3. Kernel Tracer Using SystemTap

HITACHI
Inspire the Next

- SystemTap
  - Kernel tracer infrastructure – SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live running kernel. – http://sourceware.org/systemtap/
  - Kprobes are used for probing the live running kernel. No kernel recompilation is needed for probing.

HITACHI
Inspire the Next

2

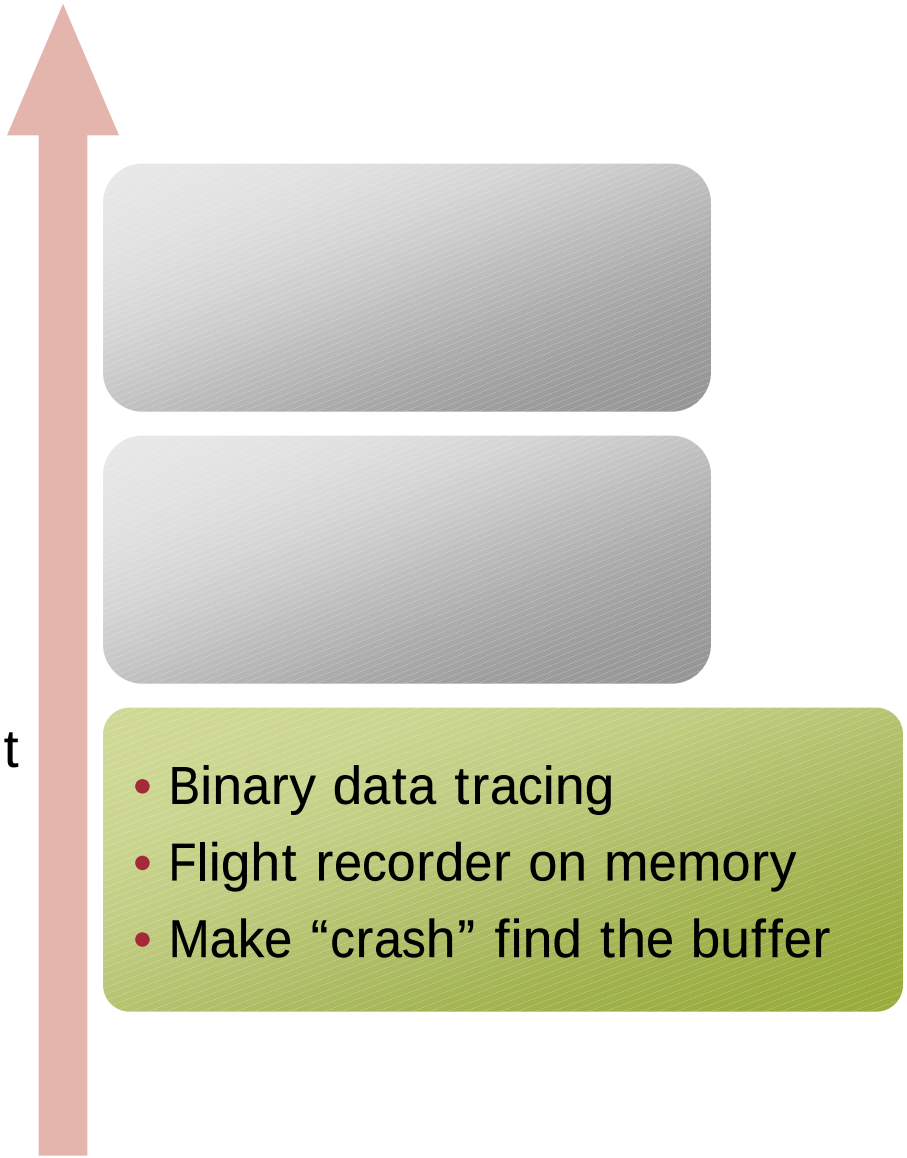
Kernel Tracer Requirement

uVALUE

2-1. SystemTap Improvement (1)

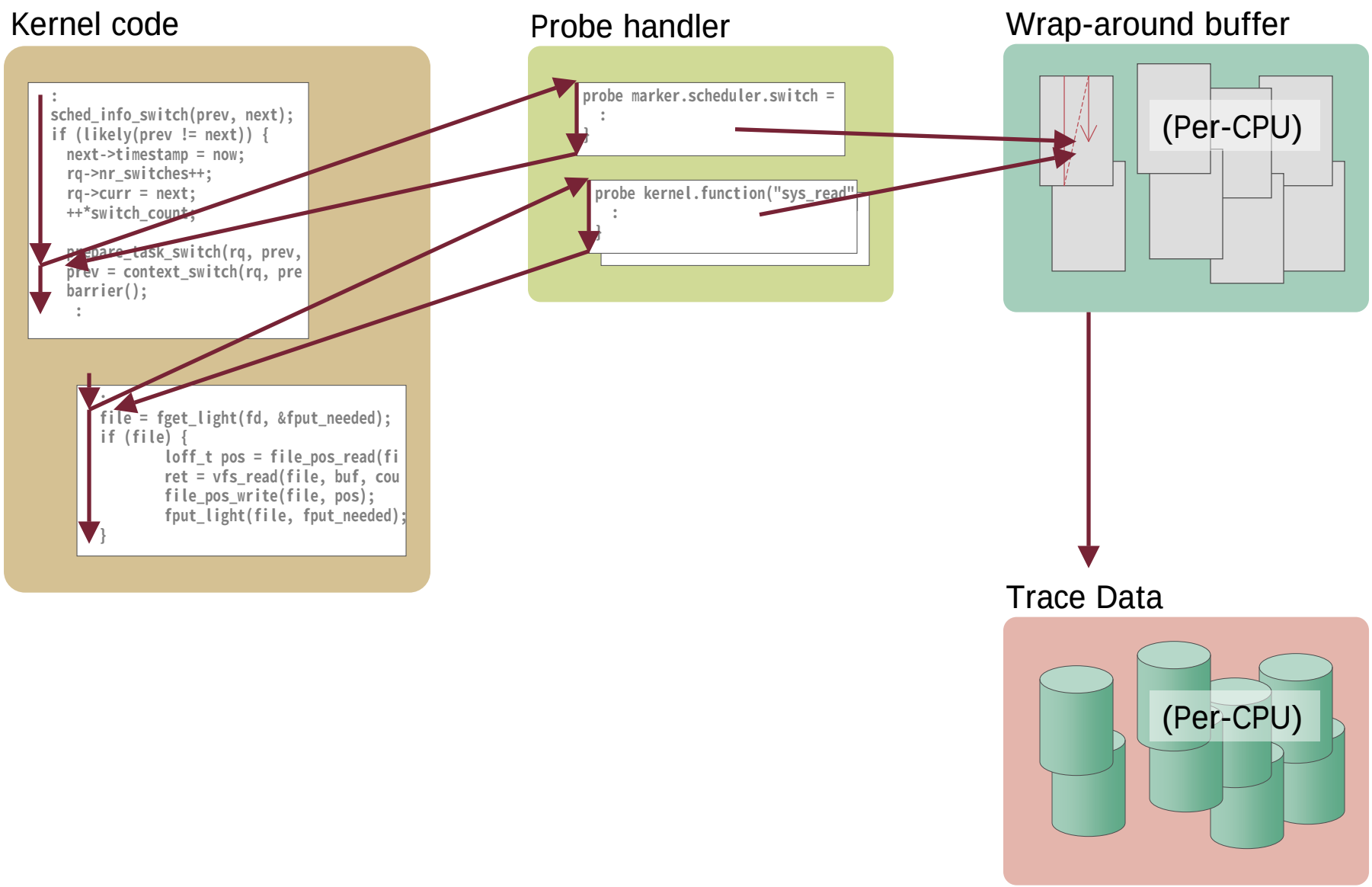
HITACHI
Inspire the Next

(1) Minimal Requirement



- Binary data tracing
- Flight recorder on memory
- Make “crash” find the buffer

2-2. How a Kernel Tracer Works



HITACHI
Inspire the Next

3

Kernel Tracer Prototype

uVALUE

3-1. Trace Points and Trace Data

HITACHI
Inspire the Next

#	Event	Trace Point	Trace Data(*)
1	System Call	Entry of each sys_... function	Syscall number and all arguments
		Exit of each sys_... function	Syscall number and return value
2	Process Switch	Entry of context_switch()	prev->prio, prev->state, next->prio, next->pid, next->comm
3	Process Wakeup	Entry of enqueue_task()	p->prio, p->state, p->pid, task_cpu(p), array->nr_active

(*) TOD (Time Of Day), CPU number, and PID are always recorded.

3-2. Sample Output

HITACHI
Inspire the Next

```
  #  |  hh:mm:ss.us      CPU    PID  |
  1  |  11:14:12.226190   #2   8877  |    sendto() -->
  2  |  11:14:12.226200   #2   8877  |    sendto() <-- 828
  3  |  11:14:12.226213   #0   8862  |  TP09[n:-5] <-- TASK_INTERRUPTIBLE
  4  |  11:14:12.226213   #0      0  |  swapper[n:20] -->
  5  |  11:14:12.226224   #4   8878  |    sendto() -->
  6  |  11:14:12.226235   #4   8878  |    sendto() <-- 828
  7  |  11:14:12.226245   #2   8877  |    futex(op = FUTEX_WAIT, val = 149941, ...) -->
  8  |  11:14:12.226249   #2   8877  |  TP09[n:-4] <-- TASK_INTERRUPTIBLE
  9  |  11:14:12.226249   #2      0  |  swapper[n:20] -->
 10  |  11:14:12.226267   #3      0  |  swapper[n:20] <-- TASK_RUNNING
 11  |  11:14:12.226279   #3      0  |  swapper[n:20] -->
 12  |  11:14:12.226280   #4   8878  |    futex(op = FUTEX_WAIT, val = 149942, ...) -->
 13  |  11:14:12.226286   #4   8878  |  TP09[n:-4] <-- TASK_INTERRUPTIBLE
 14  |  11:14:12.226286   #4      0  |  swapper[n:20] -->
 15  |  11:14:12.226338   #7      0  |  (RUNQ) #7  8883 TP09[n:-5] qlen:0
 16  |  11:14:12.226342   #7      0  |  swapper[n:20] <-- TASK_RUNNING
 17  |  11:14:12.226342   #7   8883  |  TP09[n:-5] -->
 18  |  11:14:12.226357   #7   8883  |    recvfrom() -->
 19  |  11:14:12.226360   #7   8883  |    recvfrom() <-- 820
 20  |  11:14:12.226365   #3      0  |  (RUNQ) #0  8862 TP09[n:-5] qlen:0
 21  |  11:14:12.226374   #0      0  |  swapper[n:20] <-- TASK_RUNNING
 22  |  11:14:12.226374   #0   8862  |  TP09[n:-5] →
 23  |  11:14:12.226381   #7   8883  |    futex(op = FUTEX_WAKE_OP, val = 1, ...) →
```

3-2. Sample Output

HITACHI
Inspire the Next

#	hh:mm:ss.us	CPU	PID	
1	11:14:12.226190	#2	8877	sendto() -->
2	11:14:12.226200	#2	8877	sendto() <-- 828
3	11:14:12.226213	#0	8862	TP09[n:-5] <-- TASK_INTERRUPTIBLE
4	11:14:12.226213	#0	0	swapper[n:20] -->
5	11:14:12.226224	#4	8878	sendto() -->
6	11:14:12.226235	#4	8878	sendto() <-- 828
7	11:14:12.226245	#2	8877	futex(op = FUTEX_WAIT, val = 149941, ...) -->
8	11:14:12.226249	#2	8877	TP09[n:-4] <-- TASK_INTERRUPTIBLE
9	11:14:12.226249	#2	0	swapper[n:20] -->
10	11:14:12.226267	#3	0	swapper[n:20] <-- TASK_RUNNING
11	11:14:12.226279	#3	0	swapper[n:20] -->
12	11:14:12.226280	#4	8878	futex(op = FUTEX_WAIT, val = 149942, ...) -->
13	11:14:12.226286	#4	8878	TP09[n:-4] <-- TASK_INTERRUPTIBLE
14	11:14:12.226286	#4	0	swapper[n:20] -->
15	11:14:12.226338	#7	0	(RUNQ) #7 8883 TP09[n:-5] qlen:0
16	11:14:12.226342	#7	0	swapper[n:20] <-- TASK_RUNNING
17	11:14:12.226342	#7	8883	TP09[n:-5] -->
18	11:14:12.226357	#7	8883	recvfrom() -->
19	11:14:12.226360	#7	8883	recvfrom() <-- 820
20	11:14:12.226365	#3	0	(RUNQ) #0 8862 TP09[n:-5] qlen:0
21	11:14:12.226374	#0	0	swapper[n:20] <-- TASK_RUNNING
22	11:14:12.226374	#0	8862	TP09[n:-5] →
23	11:14:12.226381	#7	8883	futex(op = FUTEX_WAKE_OP, val = 1, ...) →

3-2. Sample Output

HITACHI
Inspire the Next

```
  #  |  hh:mm:ss.us      CPU    PID  |
  1  |  11:14:12.226190   #2   8877  |    sendto() -->
  2  |  11:14:12.226200   #2   8877  |    sendto() <-- 828
  3  |  11:14:12.226213   #0   8862  |  TP09[n:-5] <-- TASK_INTERRUPTIBLE
  4  |  11:14:12.226213   #0      0  |  swapper[n:20] -->
  5  |  11:14:12.226224   #4   8878  |    sendto() -->
  6  |  11:14:12.226235   #4   8878  |    sendto() <-- 828
  7  |  11:14:12.226245   #2   8877  |    futex(op = FUTEX_WAIT, val = 149941, ...) -->
  8  |  11:14:12.226249   #2   8877  |  TP09[n:-4] <-- TASK_INTERRUPTIBLE
  9  |  11:14:12.226249   #2      0  |  swapper[
 10  |  11:14:12.226267   #3      0  |  swapper[                SK_RUNNI
 11  |  11:14:12.226279   #3      0  |  swapper[n:    ] -->
 12  |  11:14:12.226280   #4   8878  |    futex(o    = FUTEX_WAIT, v    = 149942, ...) -->
 13  |  11:14:12.226286   #4   8878  |  TP09[n:-4] <-- TASK_INTERRUPTIBLE
 14  |  11:14:12.226286   #4      0  |  swapper[n:20] -->
 15  |  11:14:12.226338   #7         |  (RUNQ  #7  8883 TP09[n:-5] qlen:0
 16  |  11:14:12.226342   #7         |  swapp   n:20] <--  SK_RUNNING
 17  |  11:14:12.226342              |  TP0
 18  |  11:14:12.226357              |    r
 19  |  11:14:12.226360   #7   8883  |    recvfrom() <-- 820
 20  |  11:14:12.226365   #3      0  |  (RUNQ) #0  8862 TP09[n:-5] qlen:0
 21  |  11:14:12.226374   #0      0  |  swapper[n:20] <-- TASK_RUNNING
 22  |  11:14:12.226374   #0   8862  |  TP09[n:-5] →
 23  |  11:14:12.226381   #7   8883  |    futex(op = FUTEX_WAKE_OP, val = 1, ...) →
```

prev->prio

prev->state

next->pid

next->comm

next->prio

3-2. Sample Output

HITACHI
Inspire the Next

#	hh:mm:ss.us	CPU	PID	
1	11:14:12.226190	#2	8877	sendto() -->
2	11:14:12.226200	#2	8877	sendto() <-- 828
3	11:14:12.226213	#0	8862	TP09[n:-5] <-- TASK_INTERRUPTIBLE
4	11:14:12.226213	#0	0	swapper[n:20] -->
5	11:14:12.226224	#4	8878	sendto() -->
6	11:14:12.226235	#4	8878	sendto() <-- 828
7	11:14:12.226245	#2	8877	futex(op = FUTEX_WAIT, val = 149941, ...) -->
8	11:14:12.226249	#2	8877	TP09[n:-4] <-- TASK_INTERRUPTIBLE
9	11:14:12.226249	#2	0	swapper[n:20] -->
10	11:14:12.226267	#3	0	swapper[n:20] <-- TASK_RUNNING
11	11:14:12.226279	#3	0	swapper[n:20] -->
12	11:14:12.226280	#4	8878	futex(op = FUT = 149942, ...) -->
13	11:14:12.226286	#4	8878	TP09[n:-4] <-- TASK_INT RUPTIBLE
14	11:14:12.226286	#4	0	swapper[n:20] -->
15	11:14:12.226338	#7	0	(RUNQ) #7 8883 TP09[n:-5] qlen:0
16	11:14:12.226342	#7	0	swapper[n:20] <-- TASK_RUNNING
17	11:14:12.226342	#7	8883	TP09[n:-5] -->
18	11:14:12.226357	#7	8883	--
19	11:14:12.226360	#7	8883	recvfrom() <-- 820
20	11:14:12.226365	#3	0	(RUNQ) #0 8862 TP09[n:-5] qlen:0
21	11:14:12.226374	#0	0	swapper[n:20] <-- TASK_RUNNING
22	11:14:12.226374	#0	8862	TP09[n:-5] →
23	11:14:12.226381	#7	8883	futex(op = FUTEX_WAKE_OP, val = 1, ...) →

p->prio

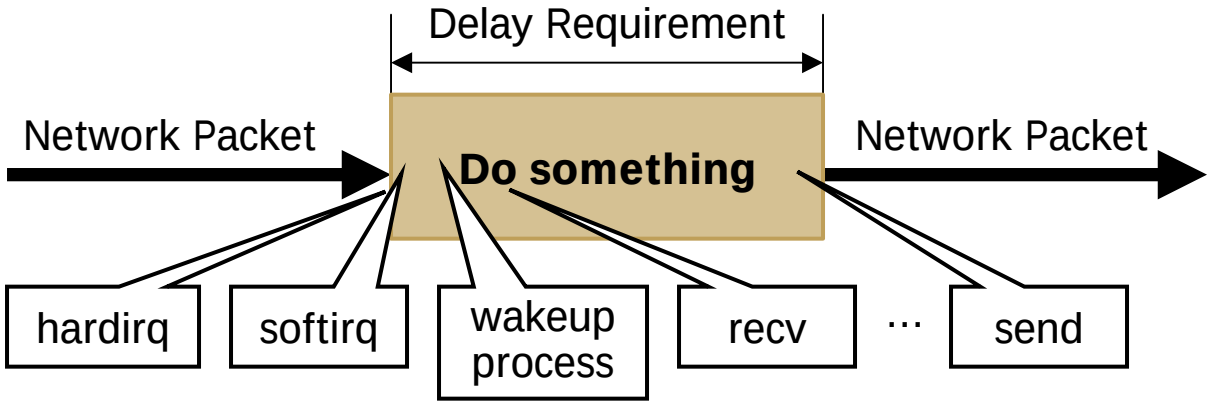
task_cpu(p)

p->pid

array->nr_active

HITACHI
Inspire the Next

- System



- Hardware
  - Intel Xeon X5460 (4 core), Dual (= 8 core)
  - 4GB of memory
- Software
  - Red Hat Enterprise Linux 5.2 (x86_64)
  - Multi-threaded application

- Problem
  - Delay (processing time) of some transactions is not satisfied.

HITACHI
Inspire the Next

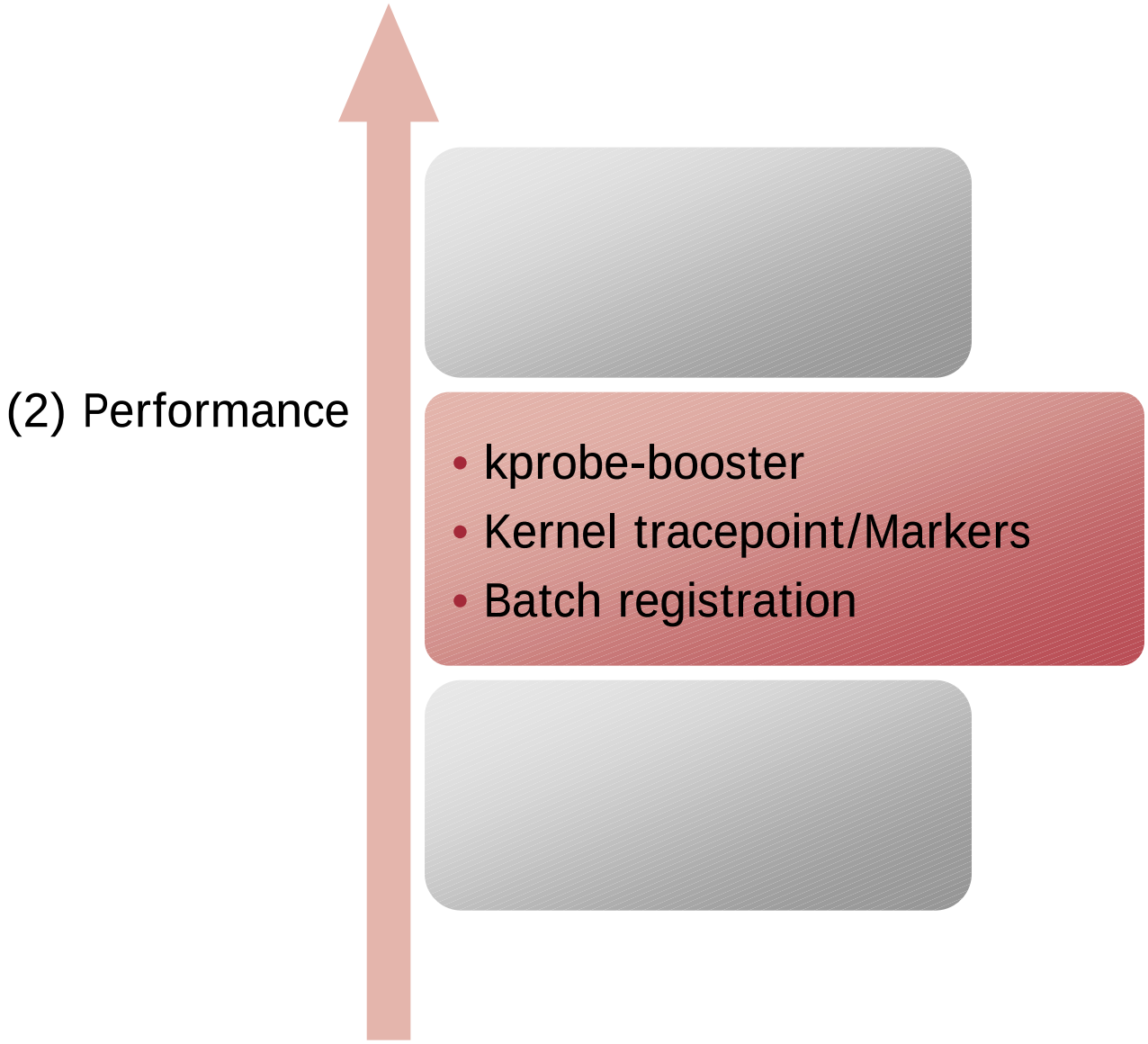
4

Kernel Tracer for Production Use

uVALUE

4-1. SystemTap Improvement (2)

HITACHI
Inspire the Next



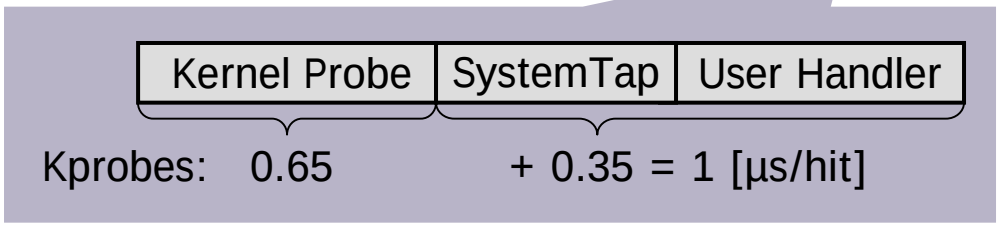
uVALUE

4-2. Runtime Overhead

HITACHI
Inspire the Next

- System-wide Overhead with Kernel Flight Recorder
  - User requirement – Less than 5% in CPU time
  - Actual number – 9% in a certain workload and script
- Breakdown of One Second (per-cpu value)
  - CPU time for trace [µs/sec] = Unit cost of trace-hit [µs/hit] × # of trace-hit [hits/sec]

90,000 [µs/sec] = 1 [µs/hit] × 90,000 [hits/sec]

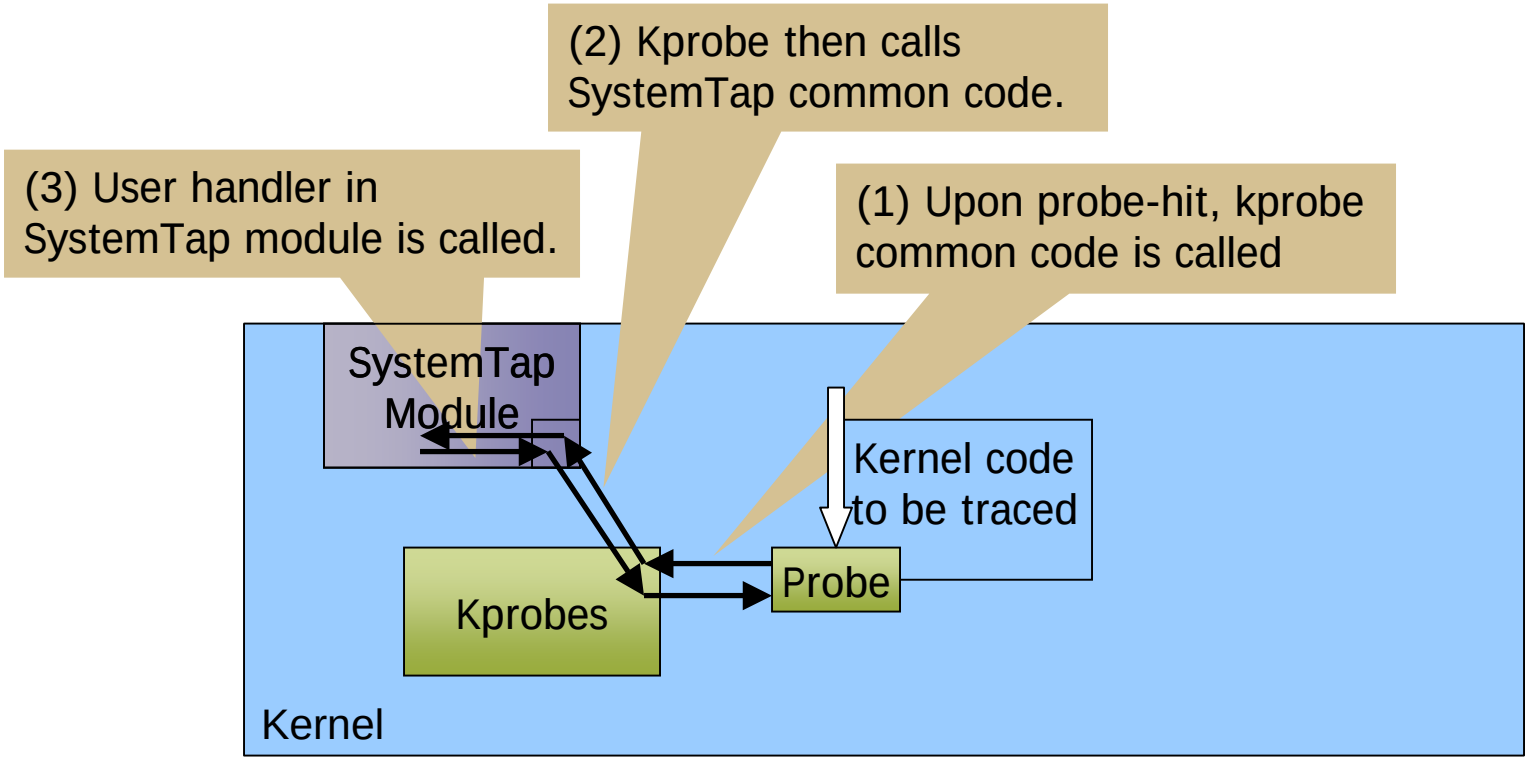


Syscall entry	25,000 [hits/sec]
Syscall exit	25,000 [hits/sec]
Context switch	5,000 [hits/sec]
Others	35,000 [hits/sec]

4-3. SystemTap Runtime Overhead

HITACHI
Inspire the Next

- Runtime Overhead on Trace-hit
  - Kernel probe code: kprobes
  - SystemTap common code (in SystemTap module)
  - User handler (in SystemTap module)

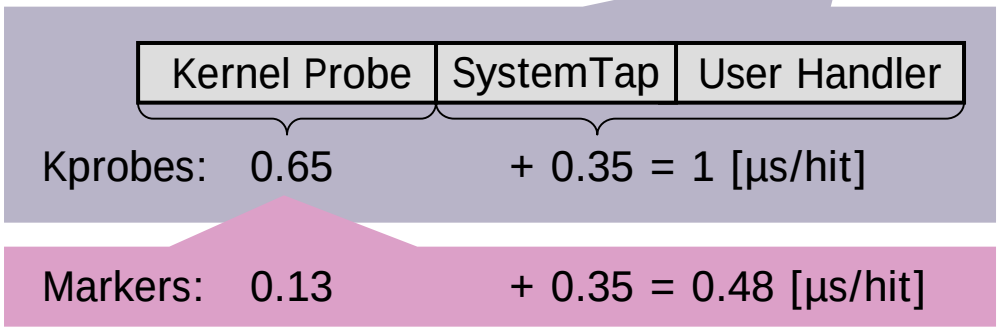


4-4. Solution with Kernel Markers

HITACHI
Inspire the Next

- System-wide Overhead with Kernel Flight Recorder
  - User requirement – Less than 5% in CPU time
  - Actual number – 9% in a certain workload and script
- Breakdown of One Second (per-cpu value)
  - CPU time for trace [µs/sec] = Unit cost of trace-hit [µs/hit] × # of trace-hit [hits/sec]

90,000 [µs/sec] = 1 [µs/hit] × 90,000 [hits/sec]



Syscall entry	25,000 [hits/sec]
Syscall exit	25,000 [hits/sec]
Context switch	5,000 [hits/sec]
Others	35,000 [hits/sec]

- Using Markers
  - If syscall entry/exit are markers, overhead will be 6.4% in CPU time.
  - 0.48 [µs/hit] × 50,000 [hits/sec] + 1 × 40,000 = 64,000 [µs/sec]

HITACHI
Inspire the Next

5

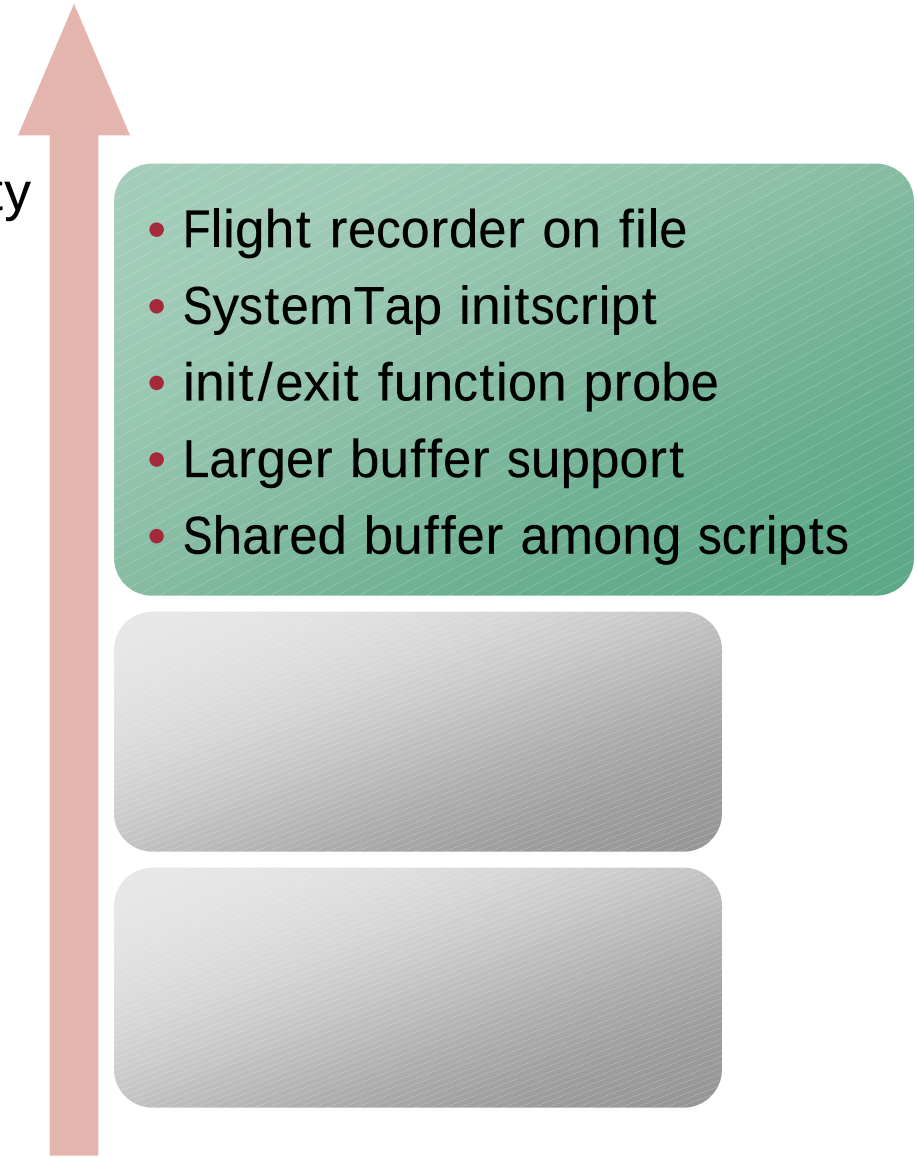
Future Work

uVALUE

5-1. SystemTap Improvement (3)

HITACHI
Inspire the Next

(3) Better Usability



5-2. Better Usability

HITACHI
Inspire the Next

- Flight Recorder on file (bz#6930(*))
  - Write the trace data on file, “rotating” at the specified size.
- SystemTap initscript (bz#6936)
  - Provide the standard way to install SystemTap scripts to a system.
  - A script can be configured to start on boot.
- init/exit function probe (bz#6503)
  - Probe init and exit function of a kernel module.
- Larger buffer support (bz#6008)
  - Support memory buffer larger than 64MB, the current limitation.
- Shared buffer among scripts (bz#3858)
  - Let script A write trace data to a buffer of script B, unlike the usual way that each script has its own buffer.

(*) http://sourceware.org/bugzilla/show_bug.cgi?id=6930 is the corresponding bugzilla entry. Change the last 4 digits to see other bugzilla entries.

5-3. Other Applications

HITACHI
Inspire the Next

- Error Path Highlighting
  - Trace the error code path and make the error remarkable, unlike Kernel Tracer that usually traces the normal (non-error) code path.
  - E.g., Print an error message in error code that is important but does not generate any error message.
- Fault Injection
  - Let the function fail by modifying kernel data (variables) or CPU registers, unlike Kernel Tracer that does not modify kernel data.
  - E.g., Let the function B return NULL at 5th call from function A.

#	Application	Code path to probe	Kernel data and registers
1	Kernel Tracer	Normal (non-error) code path	Read only
2	Error Path Highlighting	Error code path	Read only
3	Fault Injection	Normal (non-error) code path	Read and write (modify)

Trademarks

HITACHI
Inspire the Next

- Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
- Intel and Xeon are trademarks of Intel Corporation in the United States and other countries.
- Red Hat is a trademark or a registered trademark of Red Hat Inc. in the United States and other countries.

uVALUE

HITACHI

Inspire the Next