TOSHIBA
Leading Innovation >>>

Ineffective and effective way to find out latency bottlenecks by Ftrace

Yoshitake Kobayashi
Advanced Software Technology Group
Corporate Software Engineering Center
TOSHIBA CORPORATION

16 Feb, 2012

Agenda

- About Ftrace
- Problem definition
- Some actual examples to fix latency issue
- Conclusion

About Ftrace

- Ftrace is a lightweight internal tracer
  - Event tracer
  - Function tracer
  - Latency tracer
  - Stack tracer
- The trace log stay in the kernel ring buffer
- Documentation available in kernel source tree
  - Documentation/trace/ftrace.txt
  - Documentation/trace/ftrace-design.txt

About Ftrace

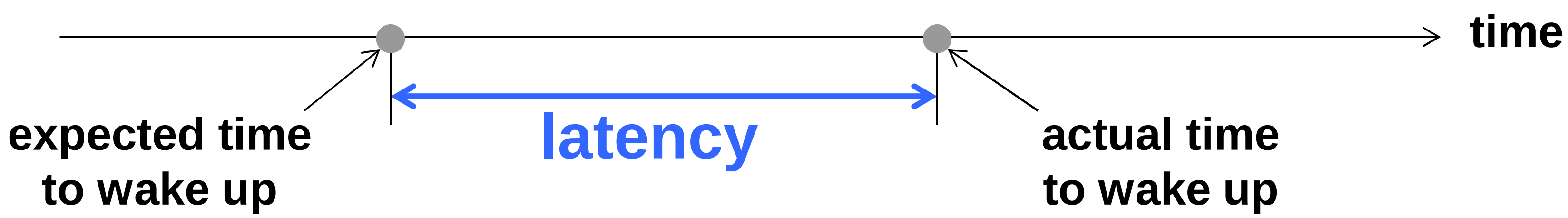
- Ftrace is a lightweight internal tracer
  - Event tracer
  - Function tracer
  - Latency tracer
  - Stack tracer
- The trace log stay in the kernel ring buffer
- Documentation available in kernel source tree
  - Documentation/trace/ftrace.txt
  - Documentation/trace/ftrace-design.txt

Definition of latency

- Latency is a measure of time delay experienced in a system, the precise definition of which depends on the system and the time being measured. Latencies may have different meaning in different contexts.

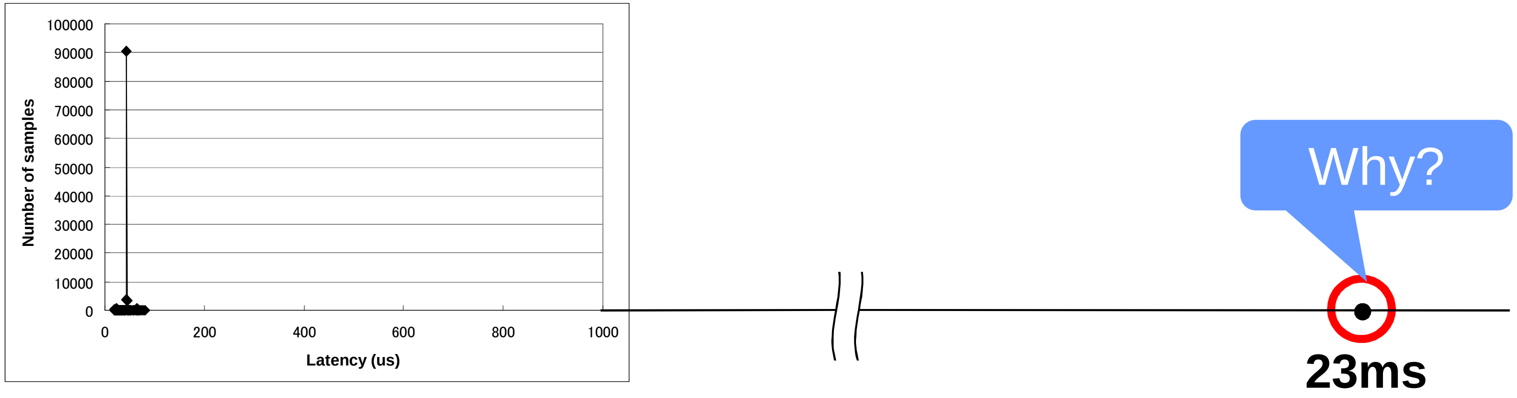
* http://en.wikipedia.org/wiki/Latency_(engineering)

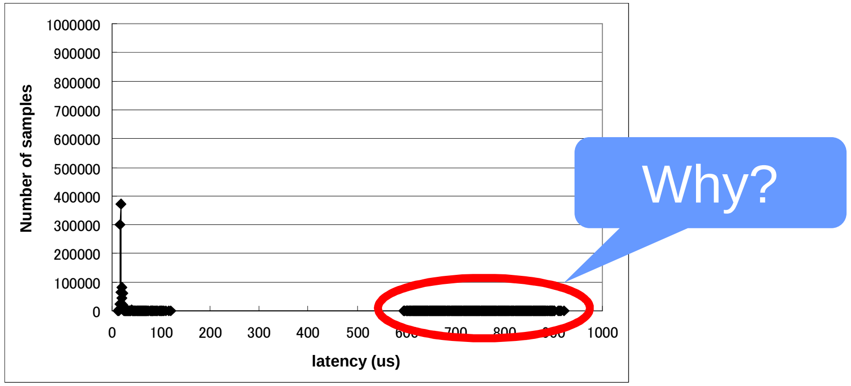
- In this presentation



Problem definition

- Evaluate the scheduling latency by Cyclictest*…. but

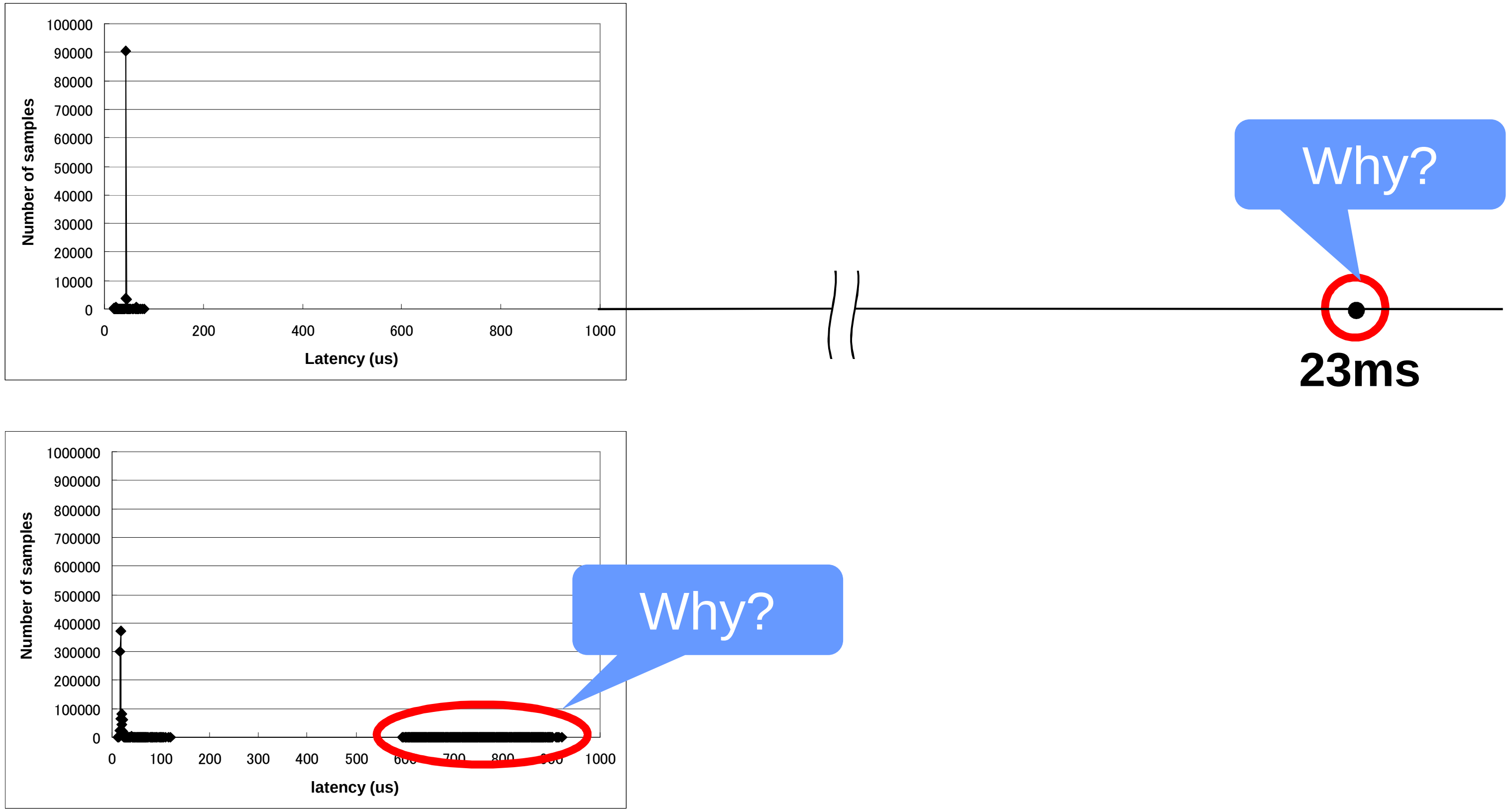




*Cyclictest (RTwiki): https://rt.wiki.kernel.org/articles/c/y/c/Cyclictest.html

Problem definition

- Evaluate the scheduling latency by Cyclictest*…. but

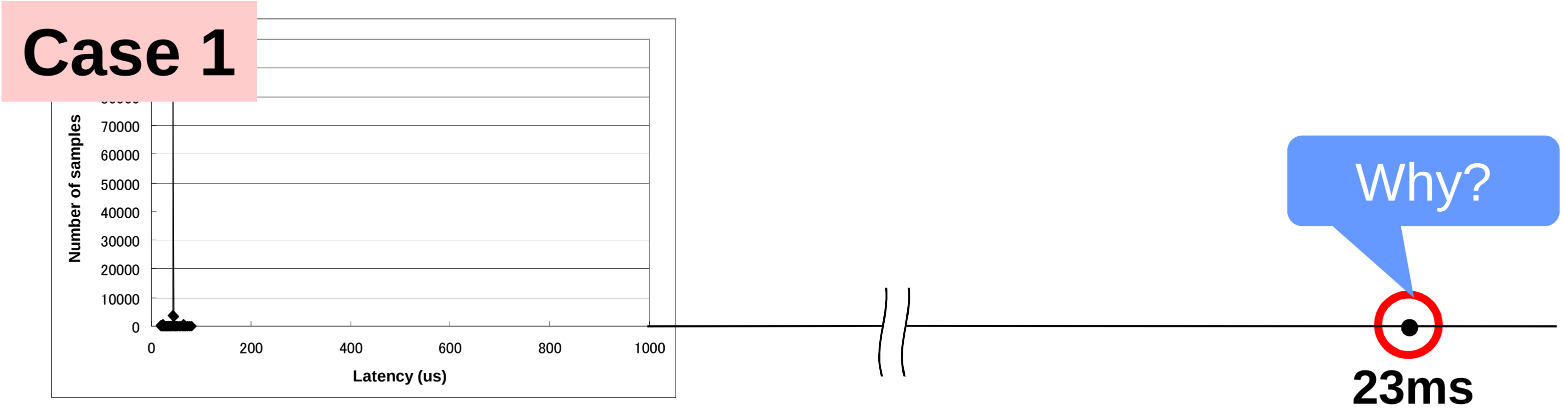


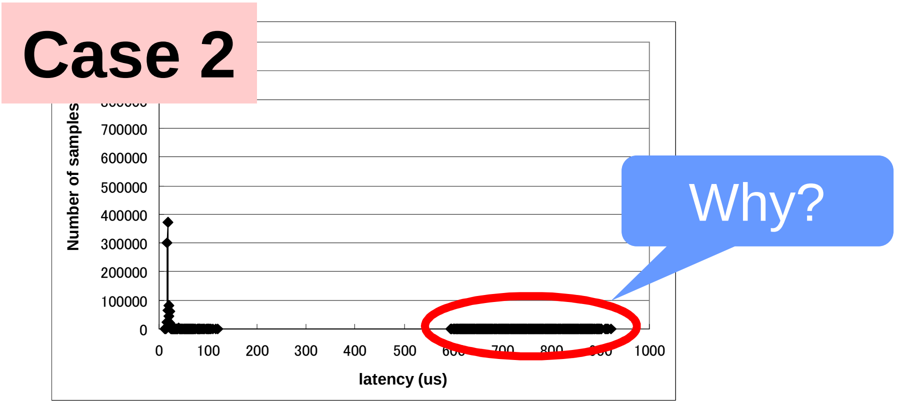
- Need to identify the cause of above problems

*Cyclictest (RTwiki): https://rt.wiki.kernel.org/articles/c/y/c/Cyclictest.html

Problem definition

- Evaluate the scheduling latency by Cyclictest*…. but





- Need to identify the cause of above issues

*Cyclictest (RTwiki): https://rt.wiki.kernel.org/articles/c/y/c/Cyclictest.html

First mistake

- Incorrect use of trace-cmd

```
# trace-cmd start –p function_graph ; target_prog ; trace-cmd stop
```

- The target_prog start with higher priority and stop if it missed deadline

First mistake

- Incorrect use of trace-cmd

```
# iostress '50%' &
# trace-cmd start –p function_graph ; target_prog ; trace-cmd stop
```

SCHED_OTHER SCHED_FIFO SCHED_OTHER

  - The target_prog start with higher priority and stop if it missed deadline
- Ftrace record the logs in the kernel ring buffer
  - The older data just overwritten by new data
  - Evaluate with too much workload is not a good idea
- Need to care about scheduling priority
  - Run a shell with higher priority
  - Stop logging in the target_prog

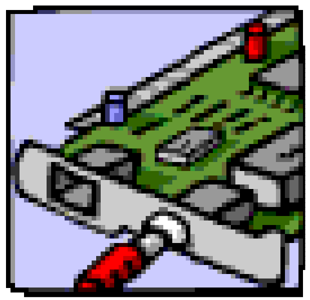
An example for stop tracing (Cyclictest)

- rt-tests/src/cyclictest/cyclictest.c
- Cyclictest has following option
  - “-b USEC” : send break trace command when latency > USEC

```
if (!stopped && tracelimit && (diff > tracelimit)) {
        stopped++;
        tracing(0);
        shutdown++;
        pthread_mutex_lock(&break_thread_id_lock);
        if (break_thread_id == 0)
                break_thread_id = stat->tid;
        break_thread_value = diff;
        pthread_mutex_unlock(&break_thread_id_lock);
}
```

Case1: Evaluation environment

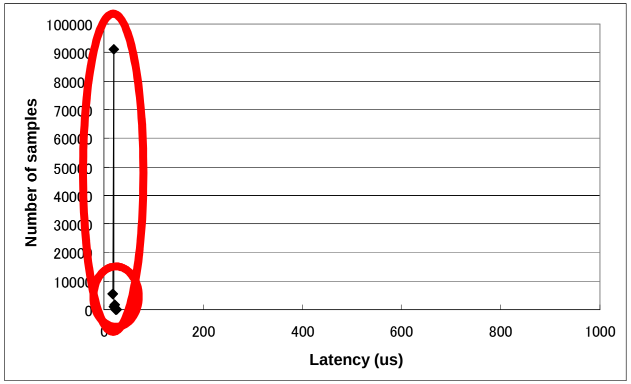
- CPU: Intel Pentium 4（2.66GHz）
- Memory: 512MB
- Kernel: Linux 2.6.31.12-rt21
  - echo -1 > /proc/sys/kernel/sched_rt_runtime_us



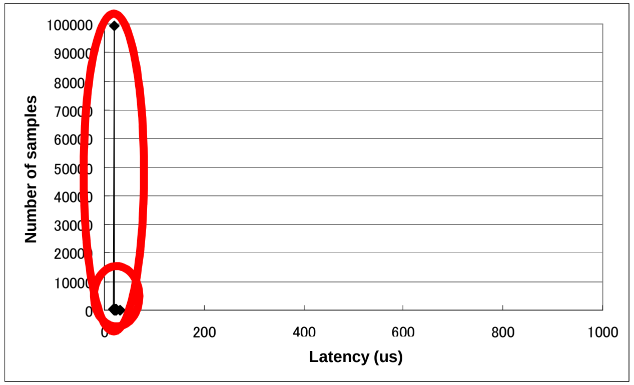
Case1: Summary of evaluation result

- Evaluated without other CPU work load

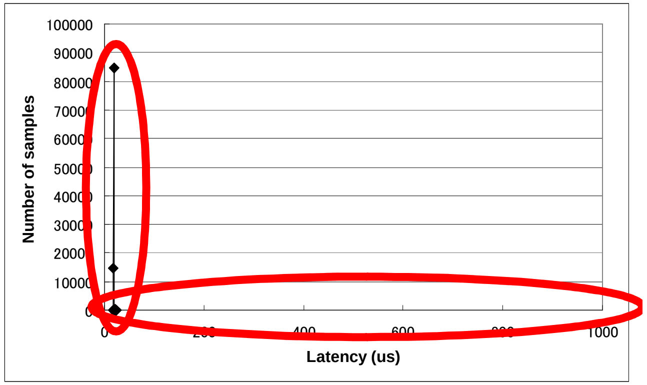
Cycle: 300us



Cycle: 500us



Cycle: 1000us



Cycle [us]	Min.	Ave.	Max	Number of DL misses
300	**18.118**	**19.162**	**25.629**	**0**
500	**18.171**	**19.269**	**32.615**	**0**
1000	**17.935**	**19.361**	**27207.563**	**1**

Case1: Check the trace log (First attempt)

- Stop the Cyclictest if the evaluated latency is higher than the given maximum
  - Backward search for sys_clock_gettime() function
  - Check the log for sys_rt_sigtimedwait() function

```
5586.207717 |   0)                |              activate_task() {
5586.207718 |   0)                |                enqueue_task() {
5586.207718 |   0)                |                  enqueue_task_rt() {
5586.207718 |   0)                |                    cpupri_set() {
5586.207719 |   0)   0.407 us     |                      _atomic_spin_lock_irqsave();
5586.207720 |   0)   0.448 us     |                      _atomic_spin_unlock_irqrestore();
5586.207721 |   0)   0.430 us     |                     _atomic_spin_lock_irqsave();
5586.207722 |   0)   0.444 us     |                      _atomic_spin_unlock_irqrestore();
5586.207723 |   0)   4.391 us     |                    }
5586.207723 |   0)   0.394 us     |                    update_rt_migration();
5586.207724 |   0)   6.158 us     |                  }
5586.207725 |   0)   7.040 us     |                }
5586.235835 |   0) ! 28117.53 us  |              }
5586.235836 |   0)                |              check_preempt_curr_rt() {
5586.235836 |   0)   0.415 us     |                resched_task();
5586.235837 |   0)   1.397 us     |              }
```

Case1: Check the kernel source (First attempt)

- The activate_task() defined in kernel/sched.c

```
static void inc_nr_running(struct rq *rq)
{
        rq->nr_running++;
}
                .
                .
                .
static void
activate_task(struct rq *rq, struct task_struct *p, int
wakeup, bool head)
{
        if (task_contributes_to_load(p))
                rq->nr_uninterruptible--;

        enqueue_task(rq, p, wakeup, head);
        inc_nr_running(rq);
}
```

Only an increment instruction

Here!

Case1: Check the trace log (Second attempt)

- Result of the function graph tracer

```
184.976213 |   0)    0.404 us    |       pre_schedule_rt();
184.976214 |   0)                |       put_prev_task_rt() {
184.976214 |   0)                |         update_curr_rt() {
185.004323 |   0)    0.537 us    |           sched_avg_update();
185.004324 |   0) ! 28109.76 us  |         }
185.004324 |   0) ! 28110.62 us  |       }
185.004325 |   0)    0.496 us    |       pick_next_task_rt();
185.004326 |   0)    0.442 us    |       perf_counter_task_sched_out();
185.004327 |   0)    0.434 us    |       native_load_sp0();
185.004328 |   0)    0.465 us    |       native_load_tls();
```

Here!

Case1: Check the kernel source (Second attempt)

- Update_curr_rt() defined in kernel/sched.c

```
static void update_curr_rt(struct rq *rq)
{
        struct task_struct *curr = rq->curr;
        struct sched_rt_entity *rt_se = &curr->rt;
        struct rt_rq *rt_rq = rt_rq_of_se(rt_se);
        u64 delta_exec;

        if (!task_has_rt_policy(curr))
                return;

        delta_exec = rq->clock - curr->se.exec_start;
        if (unlikely((s64)delta_exec < 0))
                delta_exec = 0;

        schedstat_set(curr->se.exec_max, max(curr->se.exec_max, delta_exec));

        curr->se.sum_exec_runtime += delta_exec;
        account_group_exec_runtime(curr, delta_exec);

        curr->se.exec_start = rq->clock;
        cpuacct_charge(curr, delta_exec);

        sched_rt_avg_update(rq, delta_exec);
```

Maybe here! But different result.

Case1: Check the trace log (Third attempt)

```
641.941738 |   0)              |  schedule() {
641.941738 |   0)              |    __schedule() {
641.941739 |   0)   0.421 us   |      rcu_qsctr_inc();
 . . .
641.941751 |   0)              |      pre_schedule_rt() {
641.941752 |   0)   0.418 us   |        pull_rt_task();
641.941752 |   0)   1.281 us   |      }
641.941753 |   0)              |      put_prev_task_rt() {
641.941753 |   0)              |        update_curr_rt() {
641.941754 |   0)   0.426 us   |          sched_avg_update();
641.941755 |   0)   1.310 us   |        }
641.941755 |   0)   2.178 us   |      }
641.941756 |   0)   0.423 us   |      pick_next_task_fair();
641.969863 |   0)   0.839 us   |      pick_next_task_rt();
641.969864 |   0)   0.422 us   |      pick_next_task_fair();
641.969865 |   0)   0.421 us   |      pick_next_task_idle();
641.969866 |   0)   0.461 us   |      perf_counter_task_sched_out();
```

Here!

Case1: Check the kernel source (Third attempt)

- Pick_next_task() also defined in kernel/sched.c

```
static inline struct task_struct *
pick_next_task(struct rq *rq)
{
. . .
        if (likely(rq->nr_running == rq->cfs.nr_running)) {
                p = fair_sched_class.pick_next_task(rq);
                if (likely(p))
                        return p;
        }

        class = sched_class_highest;
        for ( ; ; ) {
                p = class->pick_next_task(rq);
                if (p)
                        return p;
. . .
                class = class->next;
        }
}

asmlinkage void __sched __schedule(void)
{
. . .
        put_prev_task(rq, prev);
        next = pick_next_task(rq);
```

Different result again!

Case1: Summary of the evaluation results

- Latency occurs in different points each time and difficult to identify the cause
- Only occurs on specific hardware

↓

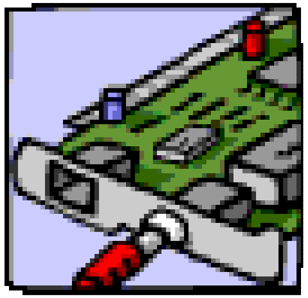
- Maybe SMI (System Management Interrupt)

Learn a lesson from Case1

- One shot trace log is not enough

Case2: Evaluation environment

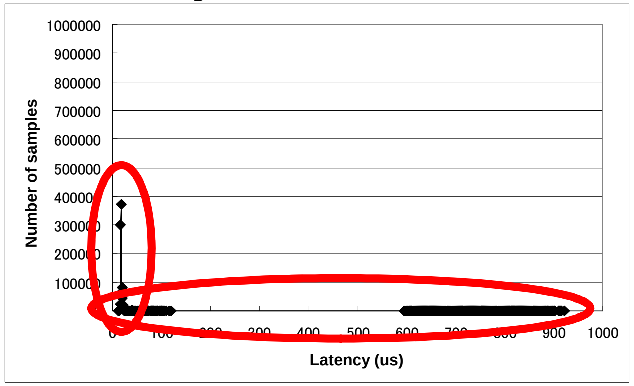
- CPU: ARM Coretex-A8
- Memory: 512MB
- Kernel: Linux 2.6.31.12-rt21
  - A function graph tracer patch applied
    http://elinux.org/Ftrace_Function_Graph_ARM



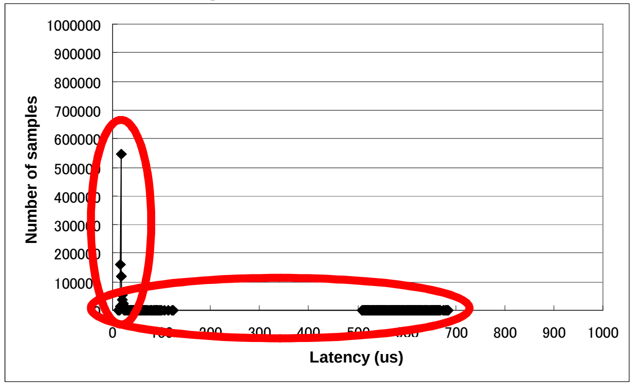
Case2: Summary of evaluation result (First attempt)

- Evaluated without other CPU workload

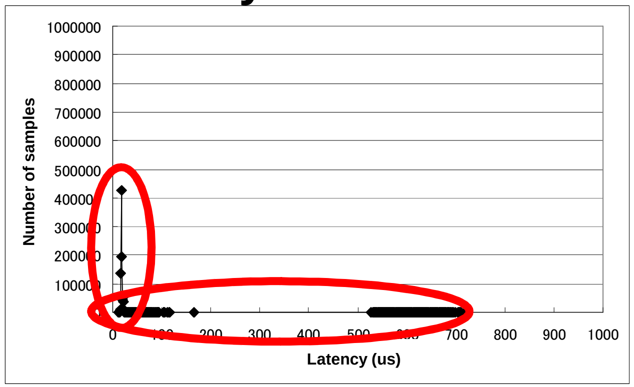
Cycle: 300us



Cycle: 500us



Cycle: 1000us

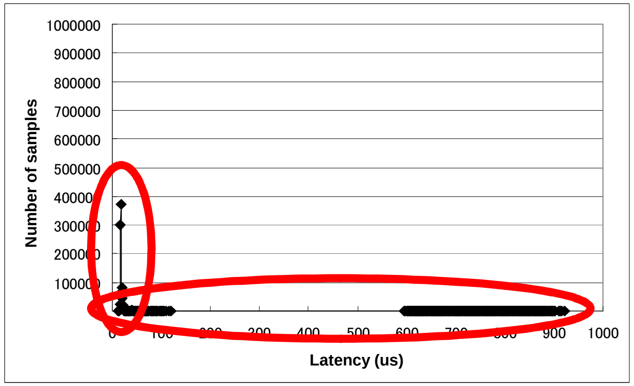


Cycle [μs]	Min.	Ave.	Max	# of DL misses
300	**11**	**19.025**	**921**	**3018**
500	**11**	**19.458**	**684**	**5026**
1000	**11**	**23.011**	**717**	**0**

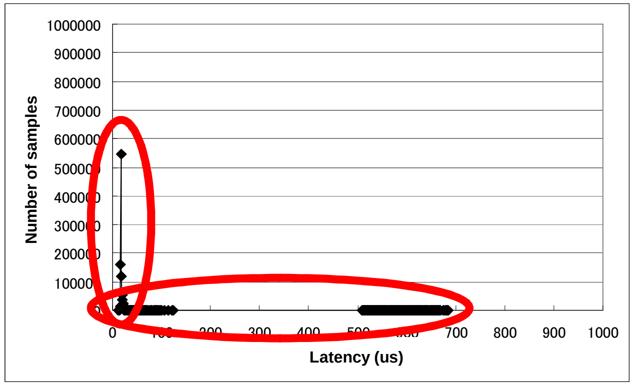
Case2: Summary of evaluation result (First attempt)

- Evaluated with approx 60% CPU workload

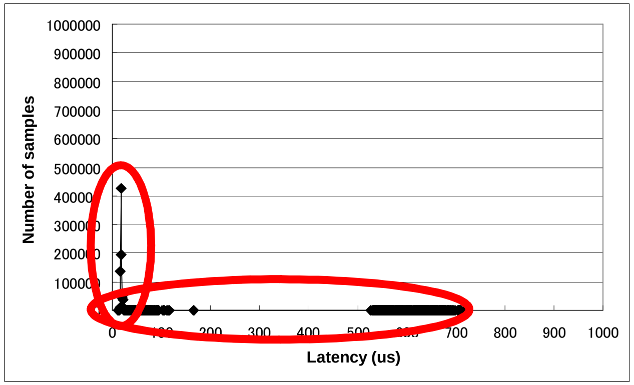
Cycle: 300us



Cycle: 500us



Cycle: 1000us



Cycle [µs]	Min.	Ave.	Max	# of DL misses
300	**13**	**34.165**	**980**	**3018**
500	**13**	**33.234**	**760**	**5025**
1000	**12**	**35.014**	**714**	**0**

Case2: Check the trace log (First attempt)

```
504.294647 |   0)              |  sys_timer_getoverrun() {
504.294649 |   0)   2.000 us   |    lock_timer();
504.294654 |   0)   6.625 us   |  }
504.294656 |   0)              |  sys_rt_sigtimedwait() {
504.294658 |   0)              |    dequeue_signal() {
504.294660 |   0)              |      __dequeue_signal() {
504.294662 |   0)   1.625 us   |        next_signal();
                       .

                              ?

                       .
504.297163 |   0)              |    recalc_sigpending() {
504.297165 |   0)              |      recalc_sigpending_tsk() {
504.297168 |   0)   5.250 us   |    }
504.297170 |   0) ! 2513.625 us |  }
504.297175 |   0)              |  sys_clock_gettime() {
504.297177 |   0)              |    posix_ktime_get_ts() {
504.297178 |   0)              |      ktime_get_ts() {
504.297181 |   0)              |        asm_do_IRQ() {
```

Here!

Case2: Identify the latency (First attempt)

```
504.295229 |   0)             |           __do_softirq() {
504.295232 |   0)             |             run_timer_softirq() {
504.295234 |   0)   1.750 us  |               hrtimer_run_pending();
504.295239 |   0)   1.750 us  |               preempt_schedule();
504.295243 |   0)             |               ehci_watchdog() {
504.295245 |   0)             |                 ehci_work() {
504.295250 |   0)   1.875 us  |                   free_cached_itd_list();
504.295256 |   0)   4.625 us  |                   qh_completions();
504.295265 |   0)   3.375 us  |                   qh_completions();
504.295272 |   0)   3.250 us  |                   qh_completions();
504.295279 |   0)   3.250 us  |                   qh_completions();
504.295286 |   0)   3.625 us  |                   qh_completions();
504.295292 |   0)   3.375 us  |                   qh_completions();
504.295299 |   0)   3.375 us  |                   qh_completions();
504.295305 |   0)   3.375 us  |                   qh_completions();
504.295312 |   0)   3.250 us  |                   qh_completions();
504.295319 |   0)   3.250 us  |                   qh_completions();
504.295325 |   0)   3.375 us  |                   qh_completions();
```

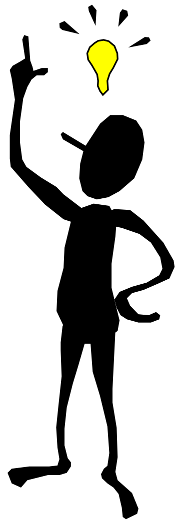
Point!

This function is used to process and free completed qtds for a qh.
(drivers/usb/host/ehci-q.c)

Case2: Cause and possible solution

- Cause
  - Too many qh_completions() function call
  - Default polling threshold is 100ms
- Possible solution
  - Change the polling threshold to 10ms

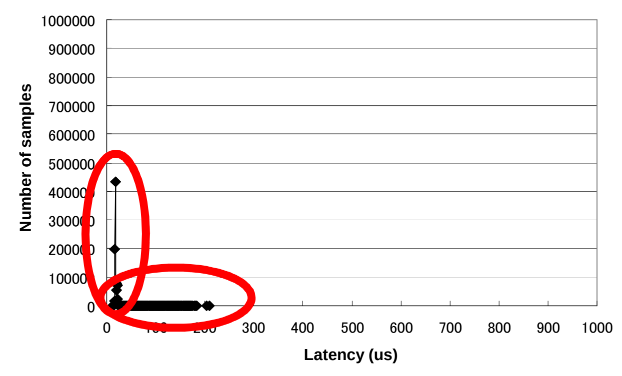
```
diff --git a/drivers/usb/host/ehci-hcd.c b/drivers/usb/host/ehci-hcd.c
index 0c9b7d2..db2efd2 100644
--- a/drivers/usb/host/ehci-hcd.c
+++ b/drivers/usb/host/ehci-hcd.c
@@ -83,7 +83,7 @@ static const char hcd_name [] = "ehci_hcd";
 #define EHCI_TUNE_FLS 2 /* (small) 256 frame schedule */
 #define EHCI_IAA_MSECS 10 /* arbitrary */
-#define EHCI_IO_JIFFIES (HZ/10) /* io watchdog > irq_thresh */
+#define EHCI_IO_JIFFIES (HZ/100) /* io watchdog > irq_thresh */
 #define EHCI_ASYNC_JIFFIES (HZ/20) /* async idle timeout */
 #define EHCI_SHRINK_FRAMES 5 /* async qh unlink delay */
```



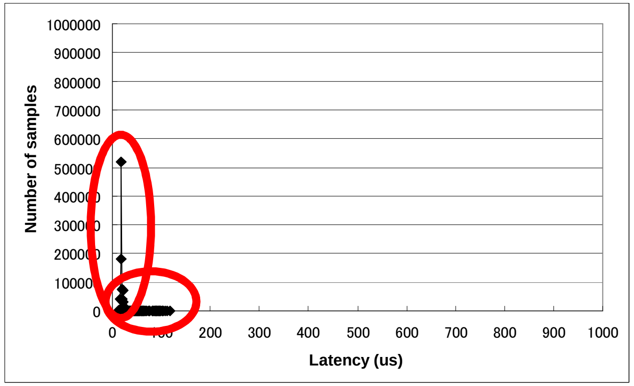
Case2: Summary of evaluation result (Second attempt)

- Evaluated without other CPU workload

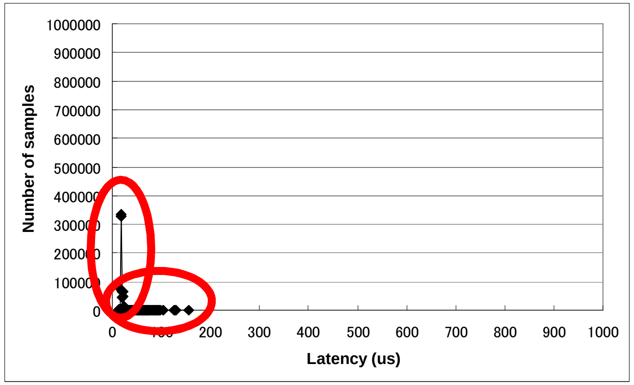
Cycle: 300us



Cycle: 500us



Cycle: 1000us

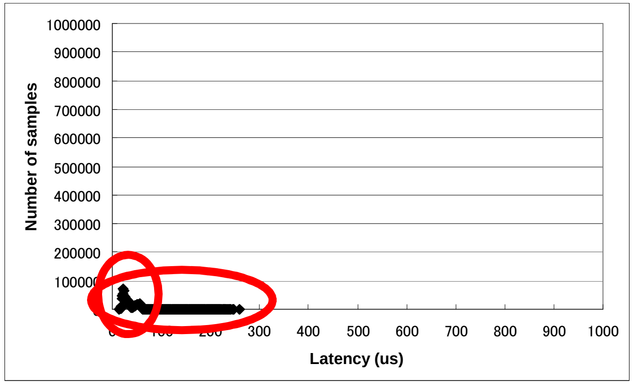


Cycle	Min	Ave.	Max	# of DL miss
300	**11**	**17.590**	**209**	**0**
500	**11**	**17.583**	**117**	**0**
1000	**11**	**17.610**	**154**	**0**

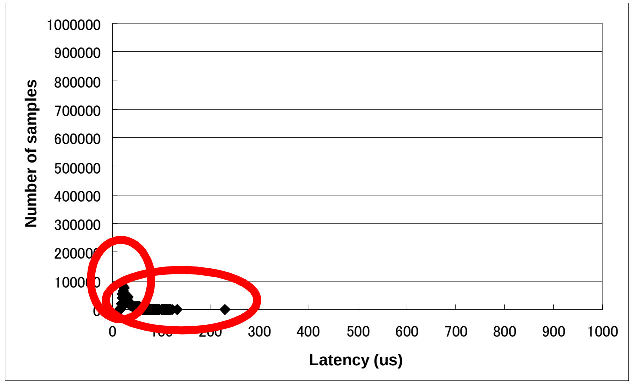
Case2: Summary of evaluation result (Second attempt)

- Evaluated with approx 60% CPU workload

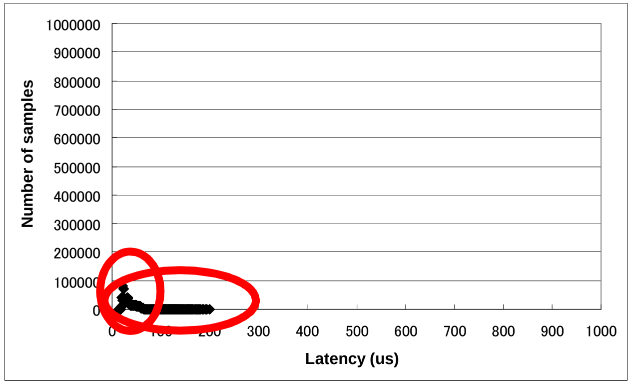
Cycle: 300us



Cycle: 500us



Cycle: 1000us



Cycle	Min	Ave	Max	# of DL miss
300	**13**	**33.365**	**259**	**0**
500	**13**	**29.695**	**228**	**0**
1000	**12**	**33.222**	**199**	**0**

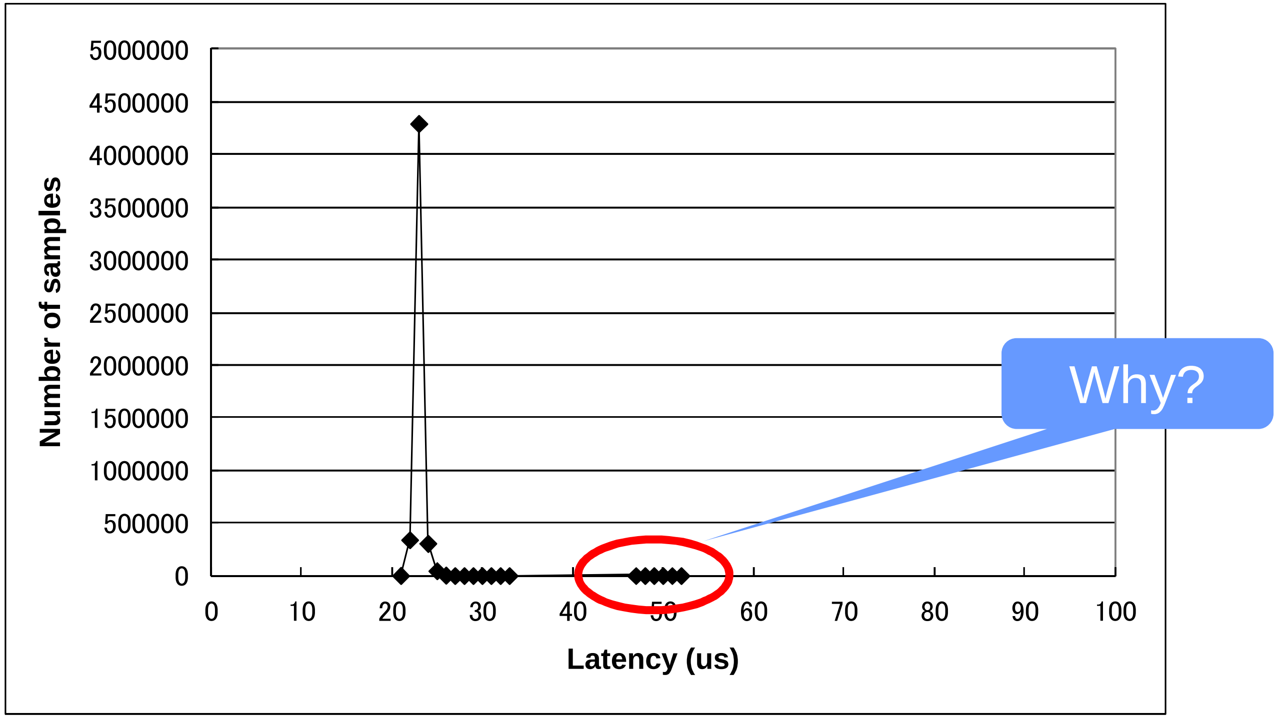
Maximum latency reduced to 1/4

Learn a lesson from Case2

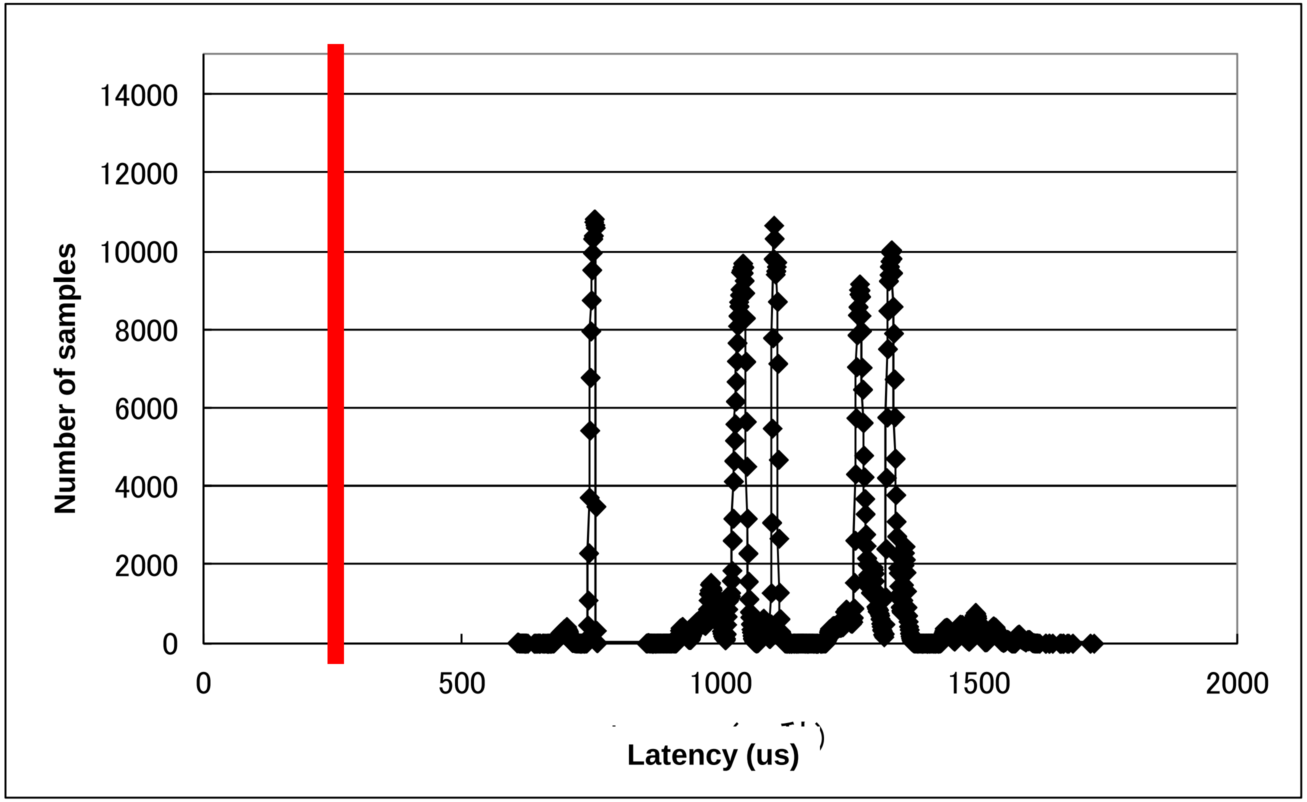
- Need to check what kind of functions are called
- Need to count the number of callees

How to stabilize latency less than 50µs

- Required spec is the following:
  - Single process
  - No network, No USB devices, No graphic device, No storage device
  - latency < 50µs
- Evaluation result (cyclictest: cycle=250µs)



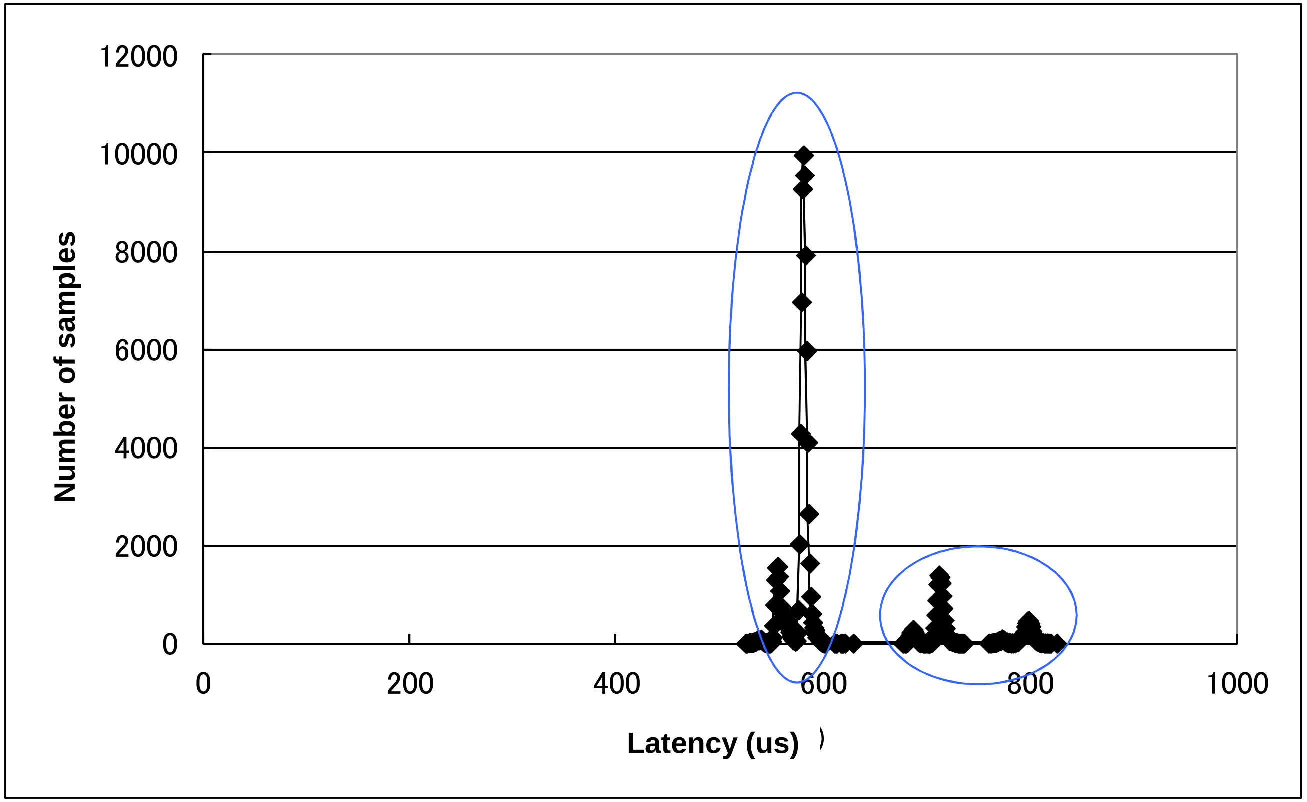
Second mistake



cycle = 250μs

- Deadline miss ratio: 100%
- Function graph tracing requires additional overhead

Evaluate the latency with trace ON



cycle = 1500µs

- This result seems to have same characteristics
- Trace with the following setting
  - Stop if the latency is more than 790µs
  - Stop if the latency is between 575 and 585

Conclusion

- Evaluating with too much workload is not a good idea
  - Required log data is lost
- One shot trace log is not enough
  - Need to check performance characteristics between trace on and off
  - Need to check if latency occurs at similar points

- Require some creative thinking to identify the cause
  - Check if the same function takes different execution time
    - Statistics approach may be applied
  - Check all functions included in specific area to identify each function’s overhead
  - Eliminate callee’s overhead to calculate pure execution time for each function’s algorithm

TOSHIBA

Leading Innovation >>>