Quality of Time Architecture & APIs

Award # CNS-1329755 (UCLA), CNS-1329644 (CMU), CNS-1329644 (UCSD), and CNS-1329650 (UCSB)
Type: Frontier; Start Date: June 2014

Fatima Anwar (UCLA), Adwait Dongare (CMU)

Co-Authors: Andrew Symington (UCLA), Mani Srivastava (UCLA), Sandeep D'souza (CMU), Anthony Rowe (CMU)

Motivation

✦**What is Quality of Time (QoT)?**

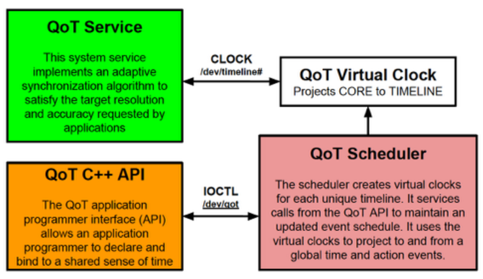
- Time is not necessarily what a clock reports. There is an uncertainty in time which is often not reported
- Quantifying this timing uncertainty with clock parameters such as accuracy, precision, jitter or wander, is what introduces quality in time

✦**Why is Quality of Time important?**

- Linux exposes few clocks e.g, *CLOCK_REALTIME, CLOCK_MONOTONIC* etc.
- These clocks are time synchronized / syntonized on best-effort basis through *NTP* or *PTP*. Thus the accuracy of these clocks is limited by underlying hardware such as, oscillators and counters
- Applications with varying demands should be able to declare their own clocks, bound to certain time bases with the desired accuracy and resolution

✦**How to control Quality of Time?**

- Produce a drift model for a local clock and map it to a declared timeline with associated quality
- Keep track of all bindings to timelines for performance optimizations



QoT Virtual Clock

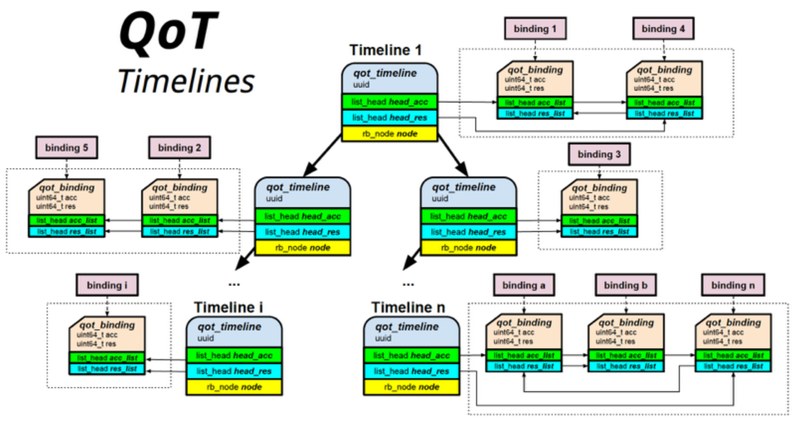
✦**Timeline:** Virtual reference time base with respect to an epoch,

- Uniquely identified by a universal identifier (***UUID***)
- Represented by a ***red black tree*** in kernel module
- Search, insertion, deletion in ***O(log(n))***

✦**Binding:** The ***accuracy*** and ***resolution*** to which a clock binds to a timeline,

- Represented by a ***linked list*** associated to a single node of red black tree
- Uniquely identified by a binding id making search, insertion and deletion of a binding in ***O(1)***

✦**Posix clock:** Timelines are exposed to the user-space in the form of posix clocks

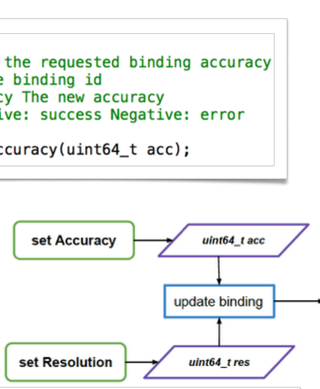


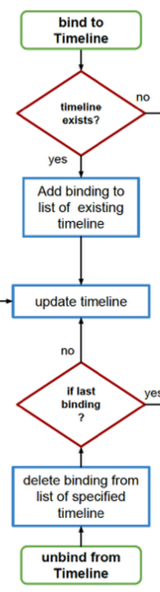

```
// Clock operations to be registred with the posix clock
static struct posix_clock_operations qot_clock_ops = {
    .owner          = THIS_MODULE,
    .clock_adjtime  = qot_clock_adjtime,    // Adjust clock both in time and frequency
    .clock_gettime  = qot_clock_gettime,    // Get current time of the clock with least uncertainty
    .clock_getres   = qot_clock_getres,     // Get resolution of the clock
    .clock_settime  = qot_clock_settime,    // Set an absolute time for the clock
    .ioctl          = qot_clock_ioctl,      // Read and write the clock metrics from and to user-space
    .open           = qot_clock_open,       // Open a handle for clock to do clock operations
    .release        = qot_clock_close,      // Release the clock handle
};
```

User APIs

```
/**
 * @brief Update the requested binding accuracy
 * @param bid The binding id
 * @param accuracy The new accuracy
 * @return Positive: success Negative: error
 **/
public: int SetAccuracy(uint64_t acc);
```

```
/**
 * @brief Update the requested binding resolution
 * @param bid The binding id
 * @param accuracy The new resolution
 * @return Positive: success Negative: error
 **/
public: int SetResolution(uint64_t res);
```



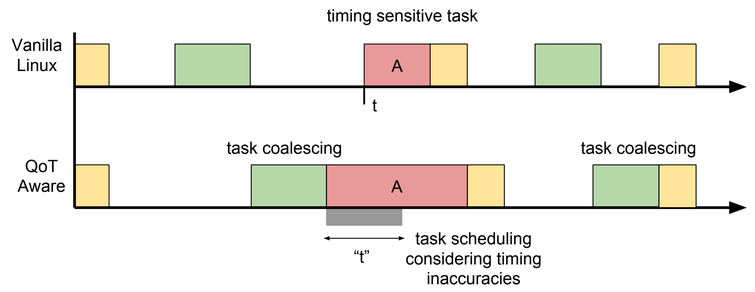



```
/**
 * @brief Request a compare action on a given pin
 * @param bid The binding id
 * @param val The global time to wakeup
 * @return < 0 : time was in the past, > 0 estimated precision
 **/
public: int64_t WaitUntil(int64_t val);
```

```
/**
 * @brief Bind to a timeline
 * @param uuid A unique identifier for the timeline
 * @param acc Required accuracy
 * @param res Required resolution
 * @return Positive: success Negative: error
 **/
public: Timeline(const std::string &uuid, uint64_t acc, uint64_t res);
```

```
/**
 * @brief Unbind from a timeline
 **/
public: ~Timeline();
```

QoT-Aware Scheduler



✦**Why QoT awareness in scheduling?**

- ***Synchronous scheduling*** of tasks at context-swap level
- Task scheduling more frequent than synchronization, causes inherent timing issues
- Identify candidates for ***task coalescing*** with rate-harmonized scheduling to improve energy efficiency

✦**Objectives**

- ***Distributed wait_until*** functionality with real-time sleep & wake-up
- Temporarily maintain ***synchronization on disconnection*** from network

Kernel Implementation

✦**Resource Kernel:** Provides ***real-time guarantees*** for RT tasks without affecting regular linux tasks

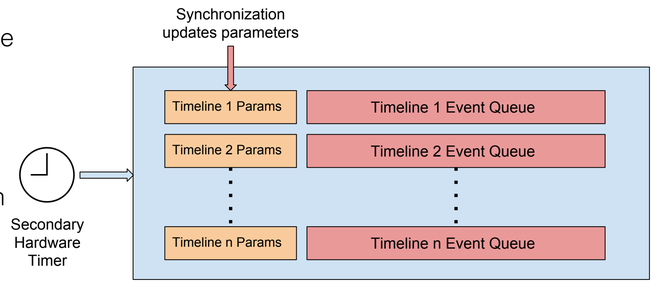
- Tasks can request guaranteed ***reservations*** on processors at particular time

✦**Timeline Queue:** Outstanding events on timeline are added to timeline queue

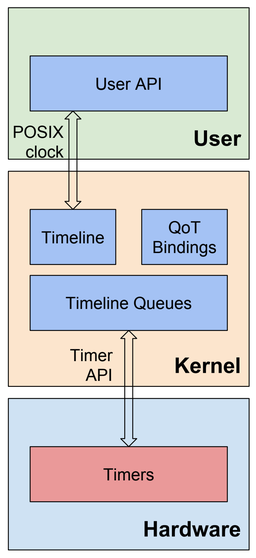
- Implemented as ***red black tree*** on per timeline basis
- Outstanding events checked on ***timer expiry*** as well as on ***changes to timeline*** (sync, adjustment, reference update)
- ***Warnings*** to tasks with missed reservations due to changing notion of time.

✦**Hardware Timer:** Timeline Queues on independent hardware timer

- Minimize interference with Linux high-resolution & system timers
- Can be referenced from ***external oscillator***



Conclusion



Introducing QoT functionality into linux through,

✦**User API:** Simple interface to interact with timelines

✦**Timelines:** Shared data structure that keeps notion of time w.r.t. compatible clocks

✦**Bindings:** Links application timing requirements with available timelines

✦**Timeline Queues:** Queue of outstanding events with checks at every time correction.

✦**Timers:** Interaction with hardware & peripheral timers with existing drivers and timer API