Virtualize Everything but Time

Timothy Broomhead (t.broomhead@ugrad.unimelb.edu.au)
Laurence Cremean (l.cremeau@ugrad.unimelb.edu.au)
Julien Ridoux (jrid@unimelb.edu.au)
Darryl Veitch (dveitch@unimelb.edu.au)
Introduction

- Clock synchronization, who cares?
  - Network monitoring / Traffic analysis
  - Telecommunications Industry; Finance; Gaming, ...
  - Distributed `scheduling': timestamps instead of message passing

- Status quo under Xen
  - Based on ntpd, amplifies its flaws
  - Fails under live VM migration

- We propose a new architecture
  - Based on RADclock client synchronization solution
  - Robust, accurate, scalable
  - Enables dependent clock paradigm
  - Seamless migration
Key Idea

- Each physical host has a single clock which never migrates

- Only a (stateless) clock read function migrates
Para-Virtualization and Xen

- **Hypervisor**
  - minimal kernel managing physical resources

- **Para-virtualization**
  - Guest OS’s have access to hypervisor via hypercalls
  - Fully-virtualized more complex, not addressed here

- **Focus on Xen**
  - But approach has general applicability!
  - Focus on Linux OS’s (2.6.31.13 Xen pvops branch)
  - Guest OS’s:
    - Dom0: privileged access to hardware devices
    - DomU: access managed by Dom0
  - Use Hypervisor 4.0 mainly
Hardware Counters

- Clocks built on local hardware (oscillators → counters)
  - HPET, ACPI, TSC
  - Counters imperfect, they drift (temperature driven)
  - Affected by OS
    ‣ ticking rate
    ‣ access latency

- TSC (counts CPU cycles)
  - Highest resolution and lowest latency - preferred! but..
  - May be unreliable
    ‣ multi-core → multiple unsynchronised TSCs
    ‣ power management → variable rate, including stopping!

- HPET
  - Reliable, but
  - Lower resolution, higher latency
Xen Clocksource

A hardware/software hybrid timer provided by the hypervisor

- **Purpose**
  - Combine reliability of HPET with low latency of TSC
  - Compensate for TSC unreliability
  - Provides 1GHz 64-bit counter

- **Performance of XCS versus HPET**
  - XCS performs well: low latency and high stability
  - HPET not that far behind, and a lot simpler
Clock Fundamentals

- Timekeeping and timestamping are distinct
- Raw timestamps and clock timestamps are distinct
- A scaled counter is not a good clock: drift!
- Purpose of clock sync algo is to correct for drift
- Network based sync is convenient, exchange timing packets:

Two key problems
- Dealing with delay variability (complex, but possible)
- Path asymmetry (simple, but impossible)
Synchronisation Algorithms

- **NTP (ntpd)**
  - Status Quo
  - Feedback based
    - Event timestamps are system clock stamps
    - Feedback controller (PLL,FLL) tries to lock onto rate
  - Intimate relationship with system clock (API, dynamics..)
  - In Xen, *ntpd* uses Xen Clocksource

- **RADclock** (Robust Absolute and Difference Clock)
  - Algo developed in 2004, extensively tested
  - Feedforward based
    - Event timestamps are raw stamps
    - Clock error estimates made and removed when clock read
  - `System clock’ has no dynamics, just a function call
  - Can use any raw counter: here use HPET, Xen Clocksource
Experimental Methodology

Internal Monitor
- Host
  - DomU
    - ntpd-NTP
    - RADclock
  - Dom0
    - ntpd-NTP
    - RADclock

External Monitor
- Host
  - Hub
  - DAG Card
    - DAG-GPS
  - Unix PC
    - UDP Sender & Receiver
  - NTP Server
    - Stratum 1
      - GPS Receiver

---
- PPS Sync.
- NTP flow
- UDP flow
- Timestamping
Wots the problem? *ntpd* can perform well

- **Ideal Setup**
  - Quality Stratum-1 time-server
  - Client is on the same LAN, lightly loaded, barely any traffic
  - Constrained and small polling period: 16 sec
Or less well...

- Different configuration (**ntpd recommended!**)
  - Multiple servers
  - Relax constraint on polling period
  - Still no load, no traffic, high quality servers

When/Why? Loss of stability a complex function of parameters ⇒ unreliable
The Xen Context

- Three examples of inadequacy of *ntpd* based solution
  1) Dependent *ntpd* clock
  2) Independent *ntpd* clock
  3) Migrating independent *ntpd* clock
1) Dependent *ntpd* Clock

- **The Solution**
  - Only Dom0 runs *ntpd*
  - Periodically updates a `boot time` variable in hypervisor
  - DomU uses Xen Clocksource to interpolate

- **The Result**  (2.6.26 kernel)
2) Independent *ntpd* Clock (current solution)

- **The Solution**
  - All guests run entirely separate *ntpd* daemons
  - Resource hungry

- **The Result**
  - When all is well, works as before but with a bit more noise
  - When works: (parallel comparison on Dom0, stratum-1 on LAN)
2) Independent \textit{ntpd} Clock (current solution)

- The Solution
  - All guests run entirely separate \textit{ntpd} daemons
  - Resource hungry

- The Result
  - Increased noise makes instability more likely
  - When fails: (DomU with some load, variable polling period, guest churn)
3) Migrating Independent \textit{ntpd} Clock

- **The Solution**
  - Independent clock as before, migrates
  - Starts talking to new system clock, new counter

- **The Result**

  \textbf{Migration Shock!}

  More Soon
RADclock Architecture

Principles

- **Timestamping:**
  - raw counter reads, not clock reads
  - independent of the clock algorithm

- **Synchronization Algorithm:**
  - based on raw timestamps and server timestamps (feedforward)
  - estimates clock parameters and makes available
  - concentrated in a single module (in userland)

- **Clock Reading**
  - combines a raw timestamp with retrieved clock parameters
  - stateless
More Concretely

- **Timestamping**
  - read chosen counter, say HPET(t)

- **Sync Algorithm maintains:**
  - **Period:** a long term average (barely changes) \(\Rightarrow\) rate stability
  - **K:** sets origin to desired timescale (e.g. UTC)
  - **E:** estimate of error \(\Rightarrow\) updates on each stamp exchange

- **Clock Reading**
  - **Absolute clock:** \(C_a(t) = \text{Period} \times \text{HPET}(t) + K - E(t)\)
    - used for absolute, and differences above critical scale
  - **Difference clock:** \(C_d(t_1,t_2) = \text{Period} \times (\text{HPET}(t_2) - \text{HPET}(t_1))\)
    - used for time differences under some critical time scale
Implementation

- **Timestamping `feedforward support`**
  - create cumulative and wide (64-bit) form of counter
  - make accessible from both kernel and user context
    - under Linux, modify Clocksource abstraction

- **Sync Algorithm**
  - Make clock parameters available via a user thread

- **Clock reading**
  - Read counter, retrieve clock data, compose
  - Fixed-point code to enable clock to be read from kernel
On Xen

Feedforward paradigm a perfect match to para-virtualisation

- **Dependent Clock now very natural**
  - Dom0 maintains a RADclock daemon, talks to timeserver
  - Makes $\text{Period}$, $\text{K}$, $\text{E}$ available through Xenstore filesystem
  - Each DomU can just reads counter, retrieve clockdata, compose

- **All Guest Clocks identically the same, but:**
  - Small delay (~1ms) in Xenstore update
    - stale data possible but very unlikely
    - small impact
  - Latency to read counter higher on DomU

- **Support Needed**
  - Expose HPET to Clocksource in guest OSs
  - Add hypercall to access platform timer (HPET here)
  - Add read/right functions to access clockdata from Xenstore
Independent RADclock on Xen

- Concurrent test on two DomU’s, separate NTP streams

![Graph showing RADclock error over time for HPET and Xen Clocksource]

- HPET Med: -2.5  IQR: 9.3
- XEN Med: 3.4  IQR: 9.5
Migration On Xen

Feedforward paradigm a perfect match to migration

- **Clocks don’t migrate, only a clock reading function does!**
  - Each Dom0 has its own RADclock daemon
  - DomU only ever calls a function, no state is migrated

- **Caveats**
  - Local copy of clockdata used to limit syscalls - needs refreshing
  - Host asymmetry will change, result in small clock jump
    ‣ asymmetry effects different for Dom0 (hence clock itself) and DomU
## Migration Comparison

### Setup
- Two machines, each Dom0 running a RADclock
- One DomU migrates with a
  - dependent RADclock
  - independent `ntpd`
Noise Overhead of Xen and Guests

The diagram shows box plots for the Round Trip Time (RTT) of the host system with different numbers of guests. The x-axis represents the number of guests (1, 2, 3, 4) and the y-axis shows the RTT in microseconds (µs).

- **Native**: The baseline shows minimal overhead.
- **Dom0**: Slight overhead compared to Native.
- **1 guest** to **4 guests**: Increasing overhead as the number of guests increases.

The lower diagram further breaks down the overhead for different guests:
- **DomU #1** to **DomU #4**: Different colors represent different guests, showing varying overhead patterns.

Overall, the overhead increases with the number of guests, indicating a trade-off between performance and resource allocation in the virtualized environment.
Noise Penalty Under C-States

- RTT Host [µs]

- Xen Clocksource
- HPET Hypervisor

- C0
- C1
- C2
- C3

- 50
- 60
- 70
- 80
- 90
- 100
- 110
Algo Performance Under C-States

![Box plot showing RADclock Error: E−median(E) [µs] for different C-states (C0, C1, C2, C3). The plot compares RADclock Xen and RADclock HPET. The data shows the distribution of RADclock errors across the different C-states.](image-url)
Conclusion

- Feed-Forward approach has many advantages
  - Difference clock defined
  - Absolute clock can be made much more robust
  - Time can be replayed
  - Simpler kernel support

- Good match to needs of para-virtualisation
  - Enables clock dependent mode that works
  - Allows seamless live migration

- RADclock project
  - Aims to replace *ntpd*
  - Client and Server code
  - Packages for FreeBSD and Linux (*Xen now supported*)