Time-based Coordination in Geo-Distributed Cyber-Physical Systems

Sandeep D’souza and Raj Rajkumar
Carnegie Mellon University

USENIX Workshop on Hot Topics in Cloud Computing ‘17
A Shared Notion of Time

- *Coordinated* Actions
- *Ordering* of Events

A Shared Notion of Time is *useful*

→ Replace *Communication* with *Local Computation*

*Liskov, Distributed Computing ‘93*
Geo-Distributed Cyber-Physical Systems

- Distributed computation, sensing and actuation
  - *coordination* at scale *(local to planetary)*
- Emerging CPS characterized by:
  - *different* applications, *same* infrastructure
  - *heterogeneous* computation and networking

A shared notion of time is useful to enable *coordinated action* in geo-distributed CPS
Coordination in Space and Time

The cloud is *key* to achieve scale

→ **Time-aware cloud for geo-distributed coordination**
CPS and the Cloud

- The cloud is key to enable geographical scaling
  - data storage
  - host the intelligence behind CPS
  - enable coordination between smaller entities
- Low-latency requirements of CPS
  - Safety-critical + real-time performance
  - A hierarchy of cloudlet and cloud deployments

Existing Research: Reduce Network Latency* and Efficient Data Storage#

Required: Time-based coordination in CPS

*Satyanarayanan, PerComm '15
#Zhang, HotCloud '15
Coordinated Vehicles using *TimeNet*

- **TimeNet**: Cyber-Physical Internet
  - *ideal* timesource, *no uncertainty*
  - *perfect* timestamping
- **Dynamic Traffic Management**
  - *city-scale* vehicular coordination
  - *time-based* hierarchical system
    - timestamps $\rightarrow$ event ordering
    - event ordering $\rightarrow$ policy

Inherent uncertainties with synchronized clocks
Outline

● Motivation
● **Background**
  ○ Quality of Time (QoT)
  ○ QoT Architecture
● The Case for Shared Time and QoT
● QoT-based Cloud CPS Architecture
● Conclusion
Quality of Time (QoT)*

- Quantified
  - using *clock parameters*:
    - accuracy, precision, drift....
  - w.r.t a *reference clock* (time)
- Each timestamp has bounds
  - Timestamp $\in [t-\varepsilon, t+\varepsilon]$
QoT Architecture*

- Caters to application timing demands
  - Applications *specify* QoT requirements
- Provides guarantees on the received QoT
  - *Tunable* clock synchronization
- Exposes the obtained timing accuracy
  - *QoT-estimation* mechanisms
- Easy-to-use, secure and scalable
  - *Robust* implementation

Applications *specify* QoT requirements, the QoT Architecture *orchestrates* the system and *returns* the delivered QoT → *closing the time loop*

*Anwar et al., RTSS ‘16*
Outline

- Motivation
- Background
  - The Case for Shared Time and QoT
    - Coordination in CPS
    - Shared Time and QoT
  - QoT-based CPS-Cloud Architecture
- Conclusion
Coordination in CPS

- Scalability
  - Both *numerical* and *geographical*
- Fault Tolerance and Reliability
  - Both *analytical* and *physical* redundancy
- Ease of Programmability
  - *coordination framework* with APIs
- Security

Need for a QoT-based coordination framework for CPS
Uncertainty: Software Systems vs CPS

- **Software Systems**
  - *lower* timing uncertainty, *better* QoT, *better* performance
    - Spanner#: *lower* uncertainty, *smaller* commit wait
- **Cyber-Physical Systems**
  - if timing uncertainty exceeds specification (*degraded QoT*)
  - system cannot operate safely

Application should be *notified* if QoT degrades
→ graceful degradation to satisfy safety requirements

#Corbett et al., OSDI ‘12
QoT-based Connected Vehicles

- Tolerable QoT Requirements based on
  - timestamps → event ordering
  - event ordering → policy

- If uncertainty exceeds tolerable limit
  - coordination policy can **adapt**
  - **Graceful Degradation:**
    - Increase vehicular spacing
  - **Safe Halt:**
    - Instruct vehicles to stop

---

Synchronized Clocks → **Scalable Coordination**
Quality of Time → **Fault Tolerance**
→ Need for a QoT-based CPS-Cloud Framework
Outline

- Motivation
- Background
- The Case for Shared Time and QoT
- **QoT-based CPS-Cloud Architecture**
  - Architectural Challenges
  - QoT Stack for Linux
- Conclusion
Architectural Challenges

- **Fault-Tolerance Support**
  - Robust QoT-estimation mechanisms
- **Global Coordination Service**
  - Distributed apps, heterogeneous infrastructure
- **Scalable Synchronization Service**
  - Tunable clock synchronization, heterogeneous communication
- **Virtualization Support**
  - Adding QoT awareness to virtualized units of computing
- **QoT-Aware Cloud Scheduling**
  - VM/container placement based on application QoT requirements

QoT-based platform-independent coordination API needed
Fault Tolerance

- Failure Scenario:
  - Clock Synchronization degrades
  - Reported QoT must degrade
- Application-specific failover mechanisms
  - Physical and Analytical Redundancy

QoT can enable fault-tolerant coordination in CPS
Enabling Coordination at Scale

- Timeline*: Virtual reference time base
- Coordinated actions, distributed components
  - all components bind to a timeline
  - each specifying its required QoT
- Required: Global-scale Timelines
  - Time-based coordination protocol

Timelines abstract away clock synchronization
→ Applications specify QoT requirements, framework orchestrates the system

*Anwar et al., RTSS ‘16
Virtualization and QoT

- Higher clock-read and interrupt latencies*
  - Can we get near-native performance?
- VM Migration*
  - Clock-related state in the VM or host?
- Delivering and exposing QoT to applications
  - Different VMs, different requirements

Virtualization support required for utilizing the cloud

*Broomhead et al., OSDI ‘10
QoT-Aware Cloud Scheduling

- Multiple virtualized units running applications
  - different QoT requirements
  - probabilistic QoT-based Service Level Agreements
- VM/container placement based on QoT requirements
  - dictate the host to which they are allocated
- Categorize Servers based on QoT rating
  - quality of on-board clocks
  - network-proximity to reference clock source

Make QoT-Aware Cloud Scheduling work with existing multi-level cloud schedulers
QoT Stack for Linux

Support for ARM and x86 platforms
open source, modular implementation, no change to the Linux kernel
Conclusion and Future Work

- Geo-Distributed CPS: “Coordination at scale”
- Using a Shared notion of Time and QoT enables:
  - Scalable Coordination with Fault Tolerance
  - Efficient Management of Time-related Resources
- QoT-based CPS-Cloud Architecture
  - Scalable Coordination and Clock Synchronization
  - Quartz-V: Adding QoT awareness to VMs
  - QoT-aware Cloud Scheduling

Synchronized Clocks $\rightarrow$ Scalable Coordination $\rightarrow$ Quality of Time $\rightarrow$ Fault Tolerance $\rightarrow$ QoT-based CPS-Cloud Coordination Framework
Thank You! Questions?
Discussion

● Adding QoT awareness to VMs
  ○ Paravirtualization
  ○ Security

● QoT-aware Cloud Scheduling
  ○ Challenges?

● Utility of QoT in Software Systems
  ○ Tracing, Databases ....

● ....