Coordinated City-Scale Traffic Management using Quartz "Time-as-a-Service"

Sandeep D'souza*, Heiko Koehler†, Akhilesh Joshi† and Ragunathan (Raj) Rajkumar*
*Carnegie Mellon University, †Nutanix Inc.
sandeepd@andrew.cmu.edu, heiko.koehler@nutanix.com, akhilesh.joshi@nutanix.com, rajkumar@andrew.cmu.edu

Abstract—**Cyber-physical systems (CPS) are increasingly interconnected and require real-time coordination among distributed entities. Real-time coordination of traffic lights at city scale to improve traffic flow and reduce trip times is one such geo-distributed application where coordination can yield significant economic and environmental benefits. While low latency is key for real-time coordination, a shared sense of time with the added notion of *Quality of Time* (QoT) is useful for fault detection, and enables fault-tolerant coordinated action in distributed CPS [1][2][3]. To enable such coordinated geo-distributed applications, we introduce Quartz, which exposes "Time-as-a-Service". Quartz features a distributed architecture, implemented using containerized micro-services, and allows applications to specify their timing requirements. Quartz orchestrates the underlying system to meet these requirements, and feeds back the delivered Quality of Time back to the application. In this work, we demonstrate the capabilities of Quartz and the utility of the notion of Quality of Time, using simulation of a distributed city-scale traffic-management system.**

I. Introduction

Time plays a key role in enabling coordination among distributed entities [4]. A non-exhaustive list of such coordinated systems includes swarm robotics [5], distributed databases [6], and city-scale traffic management. In such systems, a shared notion of time, by means of synchronized clocks, enables: (i) events to be ordered at distributed scale, and (ii) coordinated actuation to be scheduled at/by specific time instants.

Clock synchronization is a mature field and technologies such as GPS, Network Time Protocol (NTP) [7], and Precision Time Protocol (PTP) [8] have made it possible to provide distributed systems with a reliable and accurate shared notion of time. However, most of these technologies are best-effort and agnostic to application requirements. Additionally, clock synchronization is not perfect, and there is always some uncertainty in a node's estimate of the share notion of time. If this timing uncertainty exceeds an application's specifications, it can affect the quality and reliability of coordination [2].

To mitigate these issues, time needs to be exposed as a first class entity to applications. This can be done by: (i) allowing applications to specify their timing requirements (accuracy and resolution), and (ii) feeding back the delivered timing uncertainty back to the application, which enables applications to adapt in the face of timing uncertainty exceeding specified limits. Thus, fault-tolerant time-based coordination can be enabled by using the notion of *Quality of Time* (QoT) [1], which represents the uncertainty bounds corresponding to a timestamp, with respect to a clock reference. From an application perspective, if these uncertainty bounds exceed an acceptable limit, the application can enter a graceful-degradation mode, and thus be fault-tolerant in the face of clock-synchronization failure. Based on this notion of Quality of Time, [1] also introduced a reference QoT Architecture along with its corresponding implementation for Linux, called the QoT Stack for Linux.

Modern distributed cyber-physical applications are inherently complex, and consist of multiple interacting components. Thus, deploying these components and managing their life-cycle is a complex endeavor. Additionally, many of these components will be deployed in the cloud or edge devices in conjunction with other applications. In such scenarios, the use of technologies like containerization [9] simplify the deployment and life-cycle management of complex applications. Thus, we present Quartz which builds on the QoT Stack for Linux [1] to provide "Time-as-a-Service" to containerized applications. Quartz features a distributed architecture, implemented using containerized micro-services, making it easy to deploy and use across a range of platforms.

In this work, we demonstrate the capabilities of Quartz and the utility of the notion of Quality of Time, by means of a distributed city-scale traffic-management system deployed in simulation using the SUMO traffic simulator [10].

II. Quartz

Quartz is a user-space implementation of the QoT Stack for Linux, and has been built from the ground up for containerized applications. It features a rich application programming interface (API) that is centered around the notion of a *timeline* – a virtual sense of time to which applications bind with their desired accuracy interval and minimum resolution timing requirements [1]. A timeline can span multiple nodes, and provides a shared notion of time, with the desired QoT, to all distributed application components bound to it. This enables developers to easily write choreographed applications which can specify and observe timing uncertainty.

Quartz features a distributed implementation, composed of containerized services. It works by constructing a timing subsystem in Linux that runs parallel to timekeeping and POSIX timers. In this stack, quality metrics are passed alongside time calls. Quartz is still under development, and currently provides API bindings for C++ and Python applications. The following services make up the key components of Quartz:

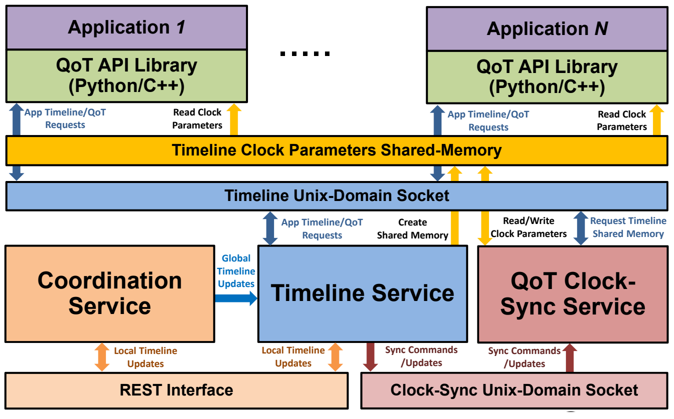
Application 1
QoT API Library (Python/C++)
.....
Application N
QoT API Library (Python/C++)
App Timeline/QoT Requests
Read Clock Parameters
App Timeline/QoT Requests
Read Clock Parameters
Timeline Clock Parameters Shared-Memory
Timeline Unix-Domain Socket
App Timeline/QoT Requests
Create Shared Memory
Read/Write Clock Parameters
Request Timeline Shared Memory
Coordination Service
Global Timeline Updates
Timeline Service
QoT Clock-Sync Service
Local Timeline Updates
Local Timeline Updates
Sync Commands /Updates
Sync Commands /Updates
REST Interface
Clock-Sync Unix-Domain Socket

Fig. 1. The Quartz Micro-Service Architecture

1) **Timeline Service** is the interface through which applications interact with Quartz, i.e., most API requests are handled by the timeline service. It is also tasked with performing the bookkeeping of the timelines that exist on a physical node, the applications bound to each timeline, and the Quality of Time requirements of each application and timeline.

2) **QoT Clock-Synchronization Service** synchronizes the per-timeline clock and computes the QoT estimates.

3) **Coordination Service** is the distributed component of the stack, responsible for discovering other nodes on a timeline, and conveying QoT requirements across nodes. This information is used by the timeline service to orchestrate the clock-synchronization service based on QoT requirements.

Figure 1 provides an overview of the different Quartz services and their interactions.

III. Demonstration Overview

In this demo, we simulate a city-scale traffic scenario with multiple intersections, using the open-source SUMO traffic simulator [10]. We use TraCI [10] to interface with the simulation, and ensure that each time-step in the simulation mirrors the flow of time in the real world. Using TraCI, we expose each intersection as MQTT endpoints which (i) periodically publish intersection state – the number of vehicles queued per-incoming lane in the last period, and (ii) listen for commands – the next phase of the traffic lights at the intersection. Note that using MQTT endpoints at the simulator interface decouples the simulation logic from the controllers.

Each intersection is controlled by an edge controller, responsible for controlling the timing and phase of the traffic lights at the intersection. Note that the controller is containerized and can be deployed on any platform. But we envision it to be placed on an edge device with a low-latency connection to the intersection. The containerized controller gets the intersection state by subscribing to the MQTT endpoints corresponding to the intersection. The controller is based on deep reinforcement-learning, which uses the current intersection state to dynamically decide the next phase of the traffic lights at the intersection. The chosen phase is published to the intersection MQTT endpoint listening for commands. Each intersection controller can also periodically receive timestamped state from

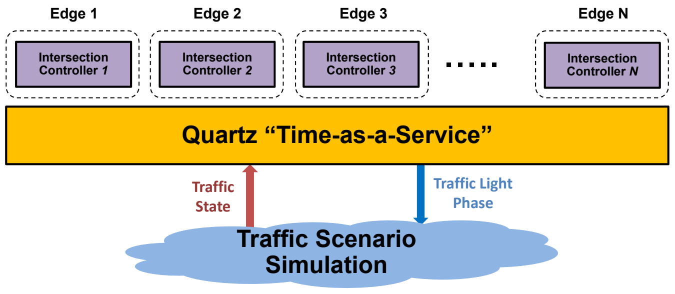
Edge 1
Edge 2
Edge 3
Edge N
Intersection Controller 1
Intersection Controller 2
Intersection Controller 3
.....
Intersection Controller N
Quartz "Time-as-a-Service"
Traffic State
Traffic Light Phase
Traffic Scenario Simulation

Fig. 2. Coordinated Traffic-Management Demo Setup

adjacent intersections, which it uses to improve traffic flow in coordination with other intersections.

In the above described scenario, a shared notion of time is key to ensure that (i) state from adjacent intersections has accurate timestamps, and (ii) the phase of the traffic lights at an intersection can be switched at an accurate time instant to ensure efficient traffic flow. Thus, each intersection controller uses Quartz to bind to the *traffic-management* timeline, while Quartz ensures that all controllers bound to the timeline share the same notion of time with the desired QoT specification. Quartz also ensures every timestamp is appended with accurate QoT estimates, enabling controllers to decide "data validity" based on the QoT bounds, i.e., data with QoT bounds beyond tolerable limits can be discarded or used with caution.

The containerized intersection controllers and the Quartz micro-services are deployed using the Nutanix Sherlock IoT platform [11]. Sherlock makes it easy to seamlessly develop, deploy, monitor and manage distributed IoT applications across multiple edge devices. Application components can be deployed as containers or event-triggered functions. Sherlock also provides the capabilities required to manage the life-cycle of applications and operationalize such systems at scale.

Figure 2 provides an overview of the architecture of the demo city-scale traffic-management application.

References

[1] F. Anwar, S. D'souza, A. Symington, A. Dongare, R. Rajkumar, A. Rowe and M. Srivastava, "Timeline: An Operating System Abstraction for Time-Aware Applications", in Proc. of *IEEE Real-Time Systems Symposium*, 2016
[2] S. D'souza and R. Rajkumar, "Time-based Coordination in Geo-Distributed Cyber-Physical Systems", in Proc. of *USENIX Workshop on Hot Topics of Cloud Computing*, 2017
[3] S. D'souza and R. Rajkumar, "QuartzV: Bringing Quality of Time to Virtual Machines", in Proc. of *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2018
[4] B. Liskov, "Practical Uses of Synchronized Clocks in Distributed Systems", in Proc. of *ACM Symposium on PODC*, 1991
[5] B. Regula, "Formation control of a large group of UAVs with safe path planning", in Proc. of *IEEE Mediterranean Conference on Control and Automation*, 2013.
[6] J. C. Corbett et al., "Spanner: Google's globally distributed database", in *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, 2013
[7] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," in *IEEE Transactions on Communication*, vol. 39, no. 10, 1991.
[8] K. Lee, J. C. Eidson, H. Weibel, and D. Mohl, "IEEE 1588-standard for a precision clock synchronization protocol for networked measurement and control systems", in *IEEE Instrumentation and Measurement Society Standard*, 2005
[9] Docker Containerization Platform, *https://www.docker.com/*
[10] Simulation of Urban Mobility (SUMO), *http://sumo.dlr.de/index.html*
[11] Nutanix IoT Platform, *https://www.nutanix.com/products/iot/*