Software-defined and Programmable CPS/IoT-OS: Architecting the Next Generation of CPS/IoT Operating Systems

Aniruddha Gokhale
Vanderbilt University
Nashville, TN
a.gokhale@vanderbilt.edu

Yogesh Barve
Vanderbilt University
Nashville, TN
yogesh.d.barve@vanderbilt.edu

Anirban Bhattacharjee
Vanderbilt University
Nashville, TN
anirban.bhattacharjee@vanderbilt.edu

Shweta Khare
Vanderbilt University
Nashville, TN
shweta.p.khare@vanderbilt.edu

ABSTRACT

The scale, heterogeneity and sources of uncertainty within modern cyber physical systems (CPS) continue to grow. For instance, a CPS may support different interaction patterns such as client-server, publish/subscribe and peer-to-peer all at once across its subsystems. Moreover, due to the uncertain environments in which they operate, a CPS often must deal with consistency-availability-partitioning (CAP) issues. Consequently, many distributed system challenges such as coordination among the different subsystems, the ability to collect real-time utilization metrics from different subsystems of the CPS, providing quality of service (QoS) assurances, and ensuring different levels of consistency guarantees must be addressed. In the existing state of the art, many of these solutions are realized using a range of third party solutions, however, this requires an enormous effort on the part of system integrators to assemble these diverse sets of frameworks and make them interoperate. To overcome these challenges, we propose that the operating systems for next generation CPS incorporate support for these capabilities natively. Moreover, these operating systems be software-defined and programmable, and architected using design patterns so that they can be programmed to self-adapt to the changing requirements and conditions. This will make it easier to design CPS that can autonomously adapt thereby improving overall system resilience.

CCS CONCEPTS

• **Computer systems organization → Distributed architectures**; **Embedded and cyber-physical systems**; • **Software and its engineering → Operating systems**; **Distributed systems organizing principles**;

KEYWORDS

CPS, Operating Systems, Software-defined, Adaptive, Customizable

NGOSCPS '19, April 15, 2019, Montreal, QC, Canada

1 INTRODUCTION

Modern cyber physical systems, such as those found in domains like smart cities and smart transportation, are no longer a single-entity CPS (e.g., a vehicle), but rather a large collection of many interacting subsystems (e.g., vehicles, road infrastructure including traffic lights, electric vehicle charging stations, and many deployed sensors such as cameras for traffic monitoring). Owing to such a large collection of interacting subsystems, there is a significant degree of heterogeneity in the types of resources used, the operating systems and middleware that execute on these resources, the programming languages used to design these subsystems, the communication interaction patterns involved among the subsystems (e.g., client-server, peer to peer, publish subscribe), and the resource management algorithms used to schedule and manage resources.

Additionally, a CPS operating environment can manifest significant variability and uncertainty in its behavior. For instance, the networks on which components internal to a vehicular subsystem communicate, e.g., CAN buses, have entirely different properties than the networks on which the vehicles in a smart transportation system communicate among themselves or to other subsystems. Similarly, the environment in which traffic sensors are deployed may manifest different behaviors and uncertainty. For instance, traffic cameras often must operate in harsh weather conditions (e.g., high winds, rain, snow) and such weather conditions will determine the image quality. Likewise, sensors embedded in roads may get damaged due to potholes causing them to stop functioning.

Given the large, distributed nature of modern CPS, and the many different and often uncertain operating environments in which they operate, these CPS will face the Consistency-Availability-Partitioning (CAP) issues [4]. The CAP theorem states that due to networks getting partitioned (often for transient periods), systems need to be designed to trade-off between maintaining strong consistency of system state versus high availability. In large systems like modern CPS, different subsystems will need to make different CAP trade-offs while ensuring that the timeliness guarantees and system correctness is maintained. Whatever the trade-offs be, this entails the need for effective coordination mechanisms (possibly hierarchical) among subsystems and their components. For instance, maintaining consistency will require consensus among subsystems. Similarly, dynamically adapting to changing conditions will require appropriate instrumentation and collection of data, which in turn

will require this data to be processed possibly centrally or across the distributed set of resources.

The present day operating systems that execute on the different entities of these CPS tend to be designed with a view to manage only a single device. Such an OS is seldom aware of the distributed environment in which the CPS operates. Although this OS may provide the basic mechanisms for internetworking and concurrency, any higher level capabilities such as CAP trade-offs, coordination, consensus, support for different interaction patterns and a whole range of many other critical distributed systems requirements must be satisfied using third party frameworks. Unfortunately, this creates a monumental challenge for system integrators who must assemble these diverse set of frameworks and make them interoperate, not to mention having to deal with differing fault models and assumptions made by each of these frameworks. This is so because the frameworks may not be designed to operate in a variety of operating environments. Second, integrating these frameworks is hard due to the differences in their interfaces, programming languages used and the associated dependencies that they must satisfy. Third, these frameworks may provide many features so as to satisfy a wide range of application requirements causing its footprint to grow. Consequently, the CPS is forced to use the entire framework even when many of its features never get used.

To address the myriad of challenges, we propose that the next generation of operating systems for CPS be designed in a way that will provide first class support within the OS itself for many of these recurring needs. Additionally, rather than creating a large footprint OS, these features should be customizable within the OS in a software-defined manner. In the remainder of this paper, we briefly describe our proposed ideas, and also provide the rationale for our thinking.

Our views on the design of next-generation OS for CPS (NGOSCPS) are informed by our recent CPS research [2, 9, 19, 20], which focuses on dynamically managing computational resources across the cloud-fog-edge spectrum to enable Cloud and IoT applications to obtain their desired quality of service (QoS) properties. In our research, we had to develop mechanisms to instrument resources for collecting their utilizations [1] and using this collected data to build system performance models that in turn can be used in autonomously managing the system-wide resources. However, to conduct such research, we had to bring together a multitude of third-party solutions, e.g., ZooKeeper [7] for coordination, Raft [15] for consensus, messaging frameworks like ZeroMQ, Kafka and AMQP, databases like InfluxDB, resource instrumentation frameworks like collectd, and many more technologies. Integrating all these solutions incurs significant efforts and is fraught with a number of accidental complexities. We surmise that other CPS researchers and practitioners are likely facing similar challenges and are required to integrate a large number of diverse, third-party frameworks to make their systems operate.

2 DESIGNING THE NEXT-GENERATION CPS OS

We call for a software-defined and programmable design of NGOSCPS. The software-defined aspect is derived from successes in software-defined networking (SDN) [11], which separates the control plane from the data plane so that the two can evolve separately, and where the intelligence is programmed into the control plane. Accordingly, we propose that NGOSCPS be designed with a separate programmable control and data plane. In the following, we explain each of the proposed attributes of NGOSCPS.

Software-defined: By making the NGOSCPS software-defined, we can push all the adaptive logic and control algorithm intelligence into the control plane. Moreover, doing so may also enable the control planes of each OS instance of a distributed set of NHOSCPS to coordinate among themselves so that they can agree upon a common set of modifications to the behaviors without needing a centralized entity to manage multiple such instances. Note that the control plane is programmable and adaptively configurable in that the desired intelligence can be programmed into the operating system's control plane. If a common set of control plane behaviors are to be deployed in multiple OS instances at the same time, it may be feasible to utilize existing cloud automation tools such as Ansible to provision such common behaviors.

The data plane is one in which the actual OS functionality manifests itself. In our case, apart from the traditional OS functions such as scheduling, memory management, etc, the data plane on each of these OS instances will comprise one or more of the third party solutions like ZooKeeper, Raft, Kafka, etc that can be customized into each instance depending on the environment in which that OS is deployed and the requirements of the application. Below we describe how such a customization may be feasible.

Docker-like packaging and customization: We propose that the NGOSCPS be customizable using an approach similar to how Docker containers are customized [12]. In the case of Docker, the customization is specified in declarative fashion in a docker file from which a Docker image is built. Often there is a base image from which the customization is made. Different customized images are usually stored in a file system, such as AUFS or OverlayFS as individual deltas so that the appropriate modules can be customized on the fly to run a Docker container. In much the same way, we propose that the customizations to the NGOSCPS be specified in a declarative manner. Such a customization should be programmable using the SDN-style control plane that we outlined above.

Unlike the case where the image of an executing Docker container cannot be changed on-the-fly, NGOSCPS must be able to adapt its behavior and packaging depending on the changes in the environment and changes in application mode of operation. The intelligence for such a dynamic adaptation should reside in the control plane. To make this a reality, we suggest that the CPS distributed environment maintain repositories of customized modules, which can be downloaded and assimilated into the OS instances on-demand. Naturally, security is an important issue, which will need further research.

Performance Interference-aware: Despite numerous advances in virtualization technologies for cloud computing and the resulting isolation among virtualized resources, co-located tenants often experience performance interference issues that degrade their performance. This happens due to the presence of non-partitionable physical resources used by these multiple tenants [10, 13]. CPS systems are also susceptible to such interference issues, and hence the NGOSCPS must be designed to be performance interference-aware [14]. By this we mean that the OS must provide capabilities to

collect utilization metrics of different resources [5] that it controls without unduly affecting the performance of that OS. The collected metrics can be analyzed to understand the degree of interference caused to the hosted tenants. The control plane can be used to configure and dynamically modify the resource instrumentation strategies.

Deployment-aware: With increasing deployment of IoT/CPS subsystems across cloud/fog/edge resources [16, 17], the NGOSCPS must be cognizant of the tier in which it is deployed. Accordingly, it must provide the capabilities to migrate tasks across the tiers, which is driven by the need to deliver QoS properties to the hosted applications [3, 22]. The actions to migrate tasks are carried by the data plane of the OS but the trigger to enforce migration is generated by the intelligence in the control plane.

Support for Data-driven Model Building and Execution: Data-driven model building and using these models to inform control actions is becoming commonplace in CPS [18, 21]. Accordingly, the NGOSCPS must also comprise capabilities to ingest data and help in distributed model building or be able to execute trained models for predictive CPS control.

Building on Prior Ideas and Standardization Issues: We believe that research ideas from the past, such as X-kernel [8] and Exokernel [6] need to be revisited in the new context where we discussed the need for fine-grained customization. We must also be cognizant of the fact that there will never be a single, common standardized OS used by CPS but as long as these OSs are designed with composability and interoperability, then many of the challenges will be overcome.

3 CONCLUSIONS

In this paper we outlined our thoughts on the design considerations for next-generation operating systems for cyber physical systems. Our ideas are informed by insights that we have gained in our recent research on dynamic resource management for IoT/CPS systems. A primary pillar of our design is the software-defined design of the OS which decouples the control plane, which is used to configure and customize the OS, from the data plane, which is used to execute the configured OS activities. The focus of our paper is on describing the non conventional ideas that we envision for such an OS. We did not focus on describing traditional OS needs for CPS, such as support for timing properties, scheduling, and other conventional OS functions. Finally, since our vision calls for pushing many of the functionalities that we find in the application-level and middleware-level frameworks into the OS, we are effectively blurring the boundaries between an OS and the middleware.

ACKNOWLEDGMENTS

This work is supported in part by NSF US Ignite CNS 1531079 and AFOSR DDDAS FA9550-18-1-0126. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF or AFOSR.

REFERENCES

[1] Yogesh Barve, Shashank Shekhar, Shweta Khare, Anirban Bhattacharjee, Zhuangwei Kang, Hongyang Sun, and Aniruddha Gokhale. 2019. FECBench: A Lightweight Interference-aware Approach for Application Performance Modeling. In *To Appear in the IEEE International Conference on Cloud Engineering (IC2E)*. Prague, Czech Republic, 10.

[2] Anirban Bhattacharjee, Ajay Dev Chhokra, Zhuangwei Kang, Hongyang Sun, and Aniruddha Gokhale. 2019. BARISTA: Efficient and Scalable Deep Learning Prediction Serving using Serverless Computing. In *To Appear in the IEEE International Conference on Cloud Engineering (IC2E)*. Prague, Czech Republic, 10.

[3] Luiz F Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F Rana, and Manish Parashar. 2017. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing* 4, 2 (2017), 26–35.

[4] Eric Brewer. 2012. CAP twelve years later: How the "rules" have changed. *Computer* 45, 2 (2012), 23–29.

[5] Christina Delimitrou and Christos Kozyrakis. 2013. ibench: Quantifying interference for datacenter applications. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE, 23–33.

[6] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole. 1995. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Symposium on Operating Systems Principles*. 251–266. citeseer.ist.psu.edu/engler95exokernel.html

[7] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems.. In *USENIX annual technical conference*, Vol. 8. Boston, MA, USA.

[8] Norman C. Hutchinson and Larry L. Peterson. 1988. Design of the *x*-Kernel. In *Proceedings of the SIGCOMM '88 Symposium*. Stanford, Calif., 65–75.

[9] Shweta Khare, Hongyang Sun, Kaiwen Zhang, Julien Gascom-Samson, Aniruddha Gokhale, and Xenofon Koutsoukos. 2018. Scalable Edge Computing Architectures for Low Latency Data Dissemination in Topic-based Publish/Subscribe. In *3rd ACM/IEEE Symposium on Edge Computing (SEC)*. Bellevue, WA, USA, 214–227.

[10] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. 2007. An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*. IEEE, 200–209.

[11] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*.

[12] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[13] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. 2010. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*. ACM, 237–250.

[14] Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. 2013. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *USENIX Conference on Annual Technical Conference (USENIX ATC'13)*. USENIX Association, Berkeley, CA, USA, 219–230. http://dl.acm.org/citation.cfm?id=2535461.2535489

[15] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 305–319.

[16] Mahadev Satyanarayanan. 2018. Edge Computing: a New Disruptive Force. Keynote Talk at the 3rd ACM/IEEE International Conference on Internet of Things Design and Implementation (IoTDI).

[17] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. 2014. Cloudlets: at the leading edge of mobile-cloud convergence. In *Mobile Computing, Applications and Services (MobiCASE), 2014 6th International Conference on*. IEEE, 1–9.

[18] Mischa Schmidt and Christer Åhlund. 2018. Smart buildings as Cyber-Physical Systems: Data-driven predictive control strategies for energy efficiency. *Renewable and Sustainable Energy Reviews* 90 (2018), 742–756.

[19] Shashank Shekhar, Hamzah Abdel Aziz, Anirban Bhattacharjee, Aniruddha Gokhale, and Xenofon Koutsoukos. 2018. Performance Interference-Aware Vertical Elasticity for Cloud-Hosted Latency-Sensitive Applications. In *IEEE International Conference on Cloud Computing (CLOUD)*. San Francisco, CA, USA, 82–89.

[20] Shashank Shekhar, Ajay Chhokra, Anirban Bhattacharjee, Guillaume Aupy, and Aniruddha Gokhale. 2017. INDICES: Exploiting Edge Resources for Performance-Aware Cloud-Hosted Services. In *IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. Madrid, Spain, 75–80. https://doi.org/10.1109/ICFEC.2017.16

[21] Rui Xiong, Fengchun Sun, Zheng Chen, and Hongwen He. 2014. A data-driven multi-scale extended Kalman filtering based parameter and state estimation approach of lithium-ion olymer battery in electric vehicles. *Applied Energy* 113 (2014), 463–476.

[22] Wuyang Zhang, Yi Hu, Yanyong Zhang, and Dipankar Raychaudhuri. 2016. SEGUE: Quality of Service Aware Edge Cloud Service Migration. In *8th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE.