# Processing Enhancement and Virtualization for Cyber-Physical Computations

Dionisio de Niz and Bjorn Andersson
SEI - Carnegie Mellon University

## ABSTRACT

Processing performance and virtualization for multitask platforms have evolved over the years in both General Purpose Computing (GPC) and Real-Time Computing (RTC). However, in RTC time has been used as a proxy to bound evolution in a physical process. This has created inefficient abstractions that are out of sync with advances in other areas of CPS such as control. In this paper we present our preliminary ideas to incorporate physics as part of the temporal clocks we use to trigger computation and provide temporal protection and virtualization.

**ACM Reference Format:**
Dionisio de Niz and Bjorn Andersson. 2019. Processing Enhancement and Virtualization for Cyber-Physical Computations. In *Proceedings of 1st, International Workshop on Next-Generation Operating Systems for Cyber-Physical Systems (NGOSCPS'19).* ACM, New York, NY, USA, 3 pages.

## 1 INTRODUCTION

The original conception of operating systems (OS) took advantage of the timeless abstraction of programming languages (ignored execution time) in general purpose (GP) computing to simplify the measure of performance to just human perception. This took the forms of throughput and average case response time as measures of performance. These measures of performance simplified the decomposition of the OS into multiple abstractions (e.g., OS layers and networked resources) whose merit had been judged with these average-case metrics. This is possible because such metrics allow the optimization of an OS component (e.g., disk driver) with some randomized generic profile (e.g., benchmarks) independently from other components that uses it (e.g., a file system). In turn, this other component is also optimized with another randomized generic profile. This strategy allowed us to create hardware-augmentation constructs that offer applications a set of simplified and generic services.

In addition to the hardware-augmentation constructs OSs have offered a shared environment to multiple applications with different levels of virtualization in the form of processes, containers, and virtual machines. These virtualizations offer strict spatial isolation and some form of temporal soft isolation focused mostly on fair-sharing of the processing time.

In contrast to GPC, in RTC the metrics of performance are not only more strict but cannot be evaluated generically for all workloads. Instead, performance needs to be verified for a specific workload. Such an evaluation takes the form of schedulability analysis. This breaks the traditional separation that allows each component in the OS to schedule their requests independently from others components. More specifically, while in real-time systems we can still let different components make independent decision on when to service a request, such an approach has a penalty. Over the years the real-time research community has been changing the way general-purpose-born OS components operate to reduce this penalty. This is the case, for instance, of disk drivers [8], networks [2], network routers [9], memory and cache allocation [12] etc.

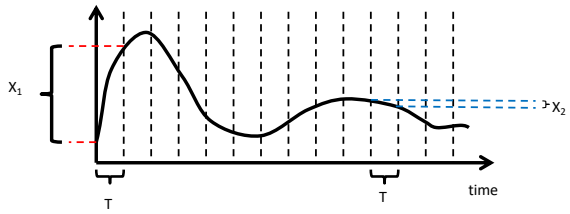


**Figure 1: Uniform Sampling**

Similar to GP OS, a real-time OS (RTOS) uses the spatial virtualization from GP OS but creates new forms of temporal virtualization that started with processing servers [10, 11] that were implemented in resource reservations [6] that had been implemented both at the kernel [7] and the hypervisor level [13].

Across all the abstractions in a traditional RTOS, the central concept that guides their design is time. Time has been used as a proxy to approximate the evolution of physical phenomena in a computer-friendly way. This matches the uniform sampling in control-theory that has served well over the years. However, this is an inefficient over-approximation that assumes that physical processes evolve at a fixed rate. For instance in the evolution of a physical variable presented in Figure 1 we can see that with uniform sampling the variable change in the first sampling ($X_1$) is much larger than the later sampling ($X_2$). The control research community has realized this and has been developing new control algorithms that calculate the sampling times based on the physical rate of change (e.g., self-triggered control [3]). In particular, self-triggered systems is a control approach where the controller calculates, based on the physical process state, both the actuation and the next time when the controller needs to actuate on the system again. Such an approach is both more efficient and more tolerant to temporal failures in networked CPS. Supporting this new trend requires thinking beyond pure time. In order to capture physics evolution more precisely we propose the creation of new clocks that incorporate the evolution of the physical processes. We call these clocks *temporal-physical clocks*. The purpose of these clocks is to improve resource allocation but also distributed coordination as we discuss in our next sections.

## 2 TEMPORAL-PHYSICAL CLOCKS FOR RESOURCE ALLOCATION

We first define a temporal-physical clock that will allow us to improve resource allocation. This clock is defined as an infinite sequence of temporal-physical *ticks* $\psi = (t, \rho)$. The temporal tick $t$ is a time duration, fixed at design time, that defines the *temporal lower-bound tick* used to upper-bound the processing demand when used as the period of the task in schedulability equations and to enforce an execution time budget. This captures the traditional minimum inter-arrival time. Then, as the task executes, it can receive at every period the *physical tick* $\rho$ that informs the scheduler the next job arrival time (relative to the current arrival) when a self-triggered-like controller is used.

### 2.1 Temporal Protection with Slack-Forseeing

A temporal-physical clock allows us to create periodic server that can foresee the slack that opens up in the future. This can be done by calculating the difference between the temporal lower-bound tick $t$ and the *physical* tick $\rho$, i.e, slack = $\rho - t$. This slack can then be used together with slack from unused reservation time to recover and schedule soft real-time tasks.

## 3 DISTRIBUTED TEMPORAL-PHYSICAL CLOCKS

We extend the temporal-physical clock to be used for distributed agreement by adding a *maximum instantaneous speed of physical change* tick $\hat{x}$ (for a physical variable $x$) to the temporal-physical tick. This allows us to extend the temporal-physical tick into the *distributed temporal-physical* tick $\delta\psi = (t, \rho, \hat{x})$ . This tick can be used to bound the maximum physical disagreement with other nodes in a distributed system. In particular, given a maximum disagreement requirement on a physical variable (e.g., space) $\vec{x}$ it is possible to calculate a maximum time $\vec{t}$ we can tolerate not exchanging agreement messages (where nodes exchange their local value $x$) that leads to a maximum amount that the local values can differ ($\vec{x}$). This time can be calculated as $\vec{t} = \frac{\vec{x}}{\hat{x}}$. This time can then be used to program a timer to trigger, say, a safe-stop action (e.g., brake the car when $x$ represents distance between two cars).

### 3.1 Disagreement Protection

The disagreement timer can be implemented in a timeout mechanism within a distributed resource reservation that can automatically trigger a safe-stop computation that stops the disagreement from growing any larger.

### 3.2 Incorporating Virtual Clocks

Clearly, the bounds on disagreement that is possible to achieve with the distributed temporal-physical clocks is coarse grained. A much finer agreement can be pursued if we incorporate virtual (logical) clocks that allows us to incorporate more events.

Virtual clocks were developed to construct an order of events happening in a distributed system (e.g., $A$ happened before $B$) [4, 5]. However, it was never the intent to capture a precise time of the event. This may be a disadvantage when capturing physical variables (reading from a sensor) that need to be merged. For instance, this is the case for sensor fusion for the same type of sensor (e.g., GPS) or different (e.g., sonar and LIDAR). In this case, merging sensing information provided by different nodes (e.g., different cars) requires having a clear understanding on when such sensing was performed. This goes beyond just an order given that the usefulness of a sensed value decreases as the physical process continues to evolve as time progresses. Some error may be tolerated but minimizing such an error is important. An alternative approach, however, would be to formulate a constraint satisfaction problem: assign values to events such that the constraints (e.g., transitivity) on virtual clocks are satisfied. This constraint satisfaction problem can be solved on each process whenever this process observes an event.

One benefit of formulating the above constraint satisfaction problem is that we can add constraints. For example, a process can typically read a local timer (real-time); we may state in the above constraint satisfaction formulation that the virtual clock value assigned to an event should be equal to the value of the local timer (real time) when this event occurred. Given that real-time clocks may not be perfectly synchronized, it may happen that the above constraint satisfaction problem is infeasible (unsatisfiable). We can address this by introducing an error for each event. The error of an event is the assigned clock value to the event minus the reading of the local real-time clock when the event occurs. Then we can formulate a cost function for all events, for example the maximum of the absolute error over all events. Then we can solve the problem: minimize the cost function subject to constraints. Once we have a solution, this approach, provides us clearly with a way to determine the order of two events and it also provides us with a timestamp of each event. Given that these timestamps minimize the error, they can be used to minimize the error in processes like sensor fusion. This approach provides an additional advantage: we can modify the cost function so that some errors are weighted more than others; this is potentially valuable because for some events it may be more import to have correspondence with the real-time clock but for others this may be less important.

Another potential benefit of formulating the above constraint satisfaction problem is that we can introduce knowledge about physics as constraints. For example, consider a ground vehicle (e.g., locomotive) that drives straight ahead over a road (or rail) and the road is partitioned into parts where for each part, there is a sensor that generates an event when the vehicle gets on that part of the road and gets off that part of the road. Then, given the physical dimension of a part of the road and the maximum speed of a vehicle, we may know a lower bound on the time of the event when the vehicle got on the part of the road and the time of the event when the vehicle got off this part of the road.

It is natural to ask whether the time required for solving this constraint satisfaction problem could be high and prohibitive for its use. Clearly, the answer to this question depends on the application. But it is noteworthy that there are already systems today that solve optimization problem for each new sample received. One example in industry today is Model-Predictive Control (MPC). Another example is the physics-based enforcer that we built in our previous research [1]. In this physics-based enforcers, we determined the satisfiability of an SMT instance whenever a controller task took a new sensor reading and it was possible to perform this computation

within a few milliseconds and in one case a bit above 10 milliseconds. The key for achieving this good performance was (i) identify some constraints that do not change and push them on the stack of the SMT solver and (ii) then at run-time, whenever a new sample is obtained, generate the specific constraints for this situation and push those. Later, when a new sensor reading is obtained, only the constraints of (ii) are popped and then new constraints of (ii) are pushed. In our previous research[1], the use of this idea provided an order of magnitude of improvement in speed for determining satisfiablity of an SMT instance. Hence, there is evidence that there may be cases where solving a constraint satisfaction for obtaining virtual-clock values of events has sufficiently low computational requirement to be useful in practice.

## 4 CONCLUDING REMARKS

We presented our preliminary ideas on temporal-physical clocks that incorporated both regular temporal clocks with physical process observations and bounds to improve resource utilization and protection. We also presented ideas on the merging of virtual clocks with physical ones and discussed how to structure solutions that can help solve CPS problems such as sensor fusion. We believe this is a rich problem to explore that can spawn new research on new abstractions and mechanisms for the new generation of CPS RTOS together with new analyses that can help us advance the state of the art in CPS.

## ACKNOWLEDGEMENT

Copyright 2019 Carnegie Mellon University. All Rights Reserved. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation. NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and "No Warranty" statements are included with all reproductions and derivative works. External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu. * These restrictions do not apply to U.S. government entities. Carnegie Mellon© is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM19-0165

## REFERENCES

[1] B. Andersson, S. Chaki, and D. de Niz. Combining symbolic runtime enforcers for cyber-physical systems. In Shuvendu Lahiri and Giles Reger, editors, *Runtime Verification*, pages 68–84, Cham, 2017. Springer International Publishing.

[2] S. Ghosh and R. R. Rajkumar. Resource management of the os network subsystem. In *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002*, pages 271–279, April 2002.

[3] W. P. M. H. Heemels, K. H. Johansson, and P. Tabuada. An introduction to event-triggered and self-triggered control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3270–3285, Dec 2012.

[4] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[5] F. Mattern. Virtual time and global states of distributed systems. In *Workshop on Parallel and Distributed Algorithms*, 1988.

[6] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: operating system support for multimedia applications. In *1994 Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 90–99, May 1994.

[7] S. Oikawa and R. Rajkumar. Portable RK: A portable resource kernel for guaranteed and enforced timing behavior. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium(RTAS)*, volume 00, page 111, 06 1999.

[8] S. Saewong and R. Rajkumar. Cooperative scheduling of multiple resources. In *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054)*, pages 90–101, Dec 1999.

[9] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pages 161–170, April 2008.

[10] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, Jun 1989.

[11] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, Jan 1995.

[12] N. Suzuki, H. Kim, D. de Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar. Coordinated bank and cache coloring for temporal protection of memory accesses. In *Proceedings of the 2013 IEEE 16th International Conference on Computational Science and Engineering*, CSE '13, pages 685–692, Washington, DC, USA, 2013. IEEE Computer Society.

[13] S. Xi, J. Wilson, C. Lu, and C. Gill. RT-Xen: Towards real-time hypervisor scheduling in Xen. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 39–48, New York, NY, USA, 2011. ACM.