A Multi-Domain Survey on Time-Criticality in Cloud Computing

R. Andreoli* R. Mini† P. Skarin† H. Gustafsson† J. Harmatos† L. Abeni* T. Cucinotta*
*Scuola Superiore Sant'Anna, Pisa, Italy, *e-mail*: {first.last}@santannapisa.it
†Ericsson Research, Lund, Sweden, *e-mail*: {first.last}@ericsson.com

***Abstract*—Conventional cloud services and infrastructures are mainly designed to maximize utilization of resources and provide best-effort Quality-of-Service levels. However, many emerging use cases in both public and private cloud computing scenarios are time-critical in nature. For example, automated vehicles, smart cities, and automated factories, are all application domains characterized by the need for highly reliable and consistent low-latency services. The incorporation of predictable execution properties in cloud solutions is essential to meet these requirements. This paper provides an overview of the current research landscape in cloud computing, summarizing the key aspects to enable support of time-critical applications. The paper explores various levels of the typical cloud software stack: machine virtualization and containers, resource management and orchestration, fault tolerance, serverless computing, data storage and management, and communications.**

***Index Terms*—Predictable Cloud Computing, Real-Time Virtualization, Resource Management and Orchestration, Real-Time Cloud Storage, Fault Tolerance, Deterministic Networking**

I. Introduction

Since the initial long-dreamed vision of computing as utility [125], cloud computing has become a disruptive technology with a steady ever-increasing impact on many sectors and businesses. The major characterizing aspects of cloud services is the illusion of infinite computing resources available on demand, and the ability to deploy horizontally scalable applications [20, 19]. These features relieve users of the burden of managing *physical infrastructure*. More recently [68], it became clear that cloud users are now facing a proliferation of *virtual resources* to be managed. This led to the widespread adoption of cloud-native applications developed using Platform-as-a-Service (PaaS) solutions, in addition to Infrastructure-as-a-Service (IaaS) ones. The latest evolution is serverless computing [23, 36, 121], where developers focus solely on writing code, leaving server provisioning and administration to the cloud provider.

The relentless growth of cloud computing has been accompanied by a steep evolution of *networking* technologies in terms of mobility and reliability. These solutions have driven the novel trend of integrating cyber-physical-systems (CPSs) and cloud infrastructures to realize the fourth stage of the industrial revolution [62], a.k.a., "Industry 4.0". Interconnected CPSs, often being mobile, Internet-of-Things (IoT) or even wearable devices, equipped with wireless and low latency networking capabilities, unlock unprecedented opportunities to develop intelligent environments: smart cities, smart healthcare and smart transportation, often enhanced with virtual and augmented reality delivered through live-streaming services [13, 14]. These use cases are all *time-critical* in nature.

Unlike conventional general-purpose use cases, the correctness of a time-critical system depends not only on the results of the computation but also on the time at which the results are produced. Time-critical systems are therefore relevant for scenarios where the inability to meet the given latency target (the so-called *deadline*) would compromise users or critical system characteristics. What distinguishes these emerging use cases is the strictness of non-functional requirements, primarily network bandwidth, latency, and reliability. Table I summarizes the four major time-critical application domains and their related emerging use cases, as identified in [13, 14, 140]: industrial control, mobility automation and robotics [22], remote control, and real-time media. For each domain, the use cases become more latency and timing critical, from top to bottom in the table, ranging from tens of milliseconds latency (i.e., low latency) to 1 millisecond (i.e., ultra-low latency). Reliability requirements can vary from 99% to 99.9999% for the most critical cases, where failure to comply with time-critical requirements may lead to catastrophic consequences (i.e. fatal accident to human personnel, damage to property, etc.). For example, in the industrial domain, a 5G/6G-enabled motion control system, which is responsible for moving and/or rotating parts of machines, requires stringent non-functional requirements, e.g., latency smaller than 2 ms and reliability greater than 99.999% [8, 146]. In the railway industry, time-critical systems controlling train movement and safety demand an end-to-end (E2E) latency smaller than 500 ms and reliability of 99.9999% [21]. Conversely, a cloud gaming service does not demand such stringent reliability, but it requires substantial network bandwidth, reaching up to 30 Mbps [14], and latencies in the 10-30ms range. Additionally, most of the use cases in remote control and real-time media can adapt their network bandwidth demand based on transient conditions.

The adoption of cloud technologies for time-critical services is hampered by performance uncertainties: general-purpose cloud-based services are mainly designed to maximize throughput and utilization of shared resources, negatively affecting predictability at run-time and resulting in best-effort Quality-of-Service (QoS) levels [48]. Predictability is essential to meet the requirements of the use cases mentioned above, so the ability for cloud platforms to meet stringent real-time and availability requirements is becoming increasingly important. This need has pushed computation and storage from

TABLE I
TIME-CRITICAL USE CASE REQUIREMENTS [13].

Domain	Use cases	Bw.	Lat.	Rel.	Adp.
Industrial control	Process monitoring	L	L	H	N
	Control to control in production line	L	L	H	N
	Programmable Logic Controller for robots	L	L	H	N
	Smart grid control	L	L	H	N
	Machine vision for robotics	H	L	H	N
	Closed-loop process control	L	UL	H	N
	Motion control	L	UL	H	N
Mobility automation	Automated container transport in port	L	L	H	N
	Machine vision for intersection safety	H	L	H	N
	Cooperative maneuvering of vehicles	L	L	H	N
	Collaborative mobile robots	L	UL	H	N
	Motion control of autonomous vehicles	L	UL	H	N
Remote control	Remote control with video/audio	H	L	O	Y
	Remote control with AR overlay	H	UL	O	Y
	Remote control with haptic feedback	H	UL	O	Y
Real-time media	Cloud-assisted basic AR	M	L	O	Y
	Premium-experience cloud-assisted AR	M	L	O	Y
	Cloud gaming	H	L	O	Y
	Interactive VR cloud gaming	H	L	O	Y
	Cloud-rendered AR	H	UL	O	Y
	Media production	H	UL	O	N

Acronyms: Virtual Reality (VR), Augmented Reality (AR); Network Bandwidth (Bw.), Latency (Lat.), Reliability (Rel.), Adaptive (Adp.); Ultra-Low (UL), Low (L), Medium (M), High (H), Optional (O).

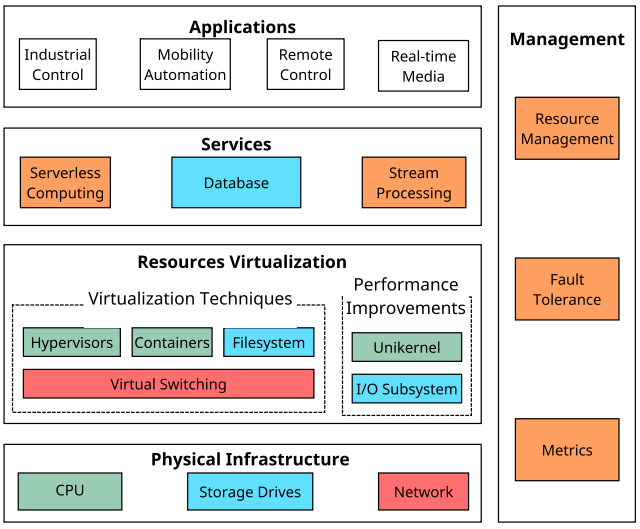


Fig. 1. Cloud computing architecture with colors representing the taxonomy used in this survey: Compute (green), Orchestration (orange), Storage (blue), and Communication (red).

centralized data centers towards the "edge" of the network. *Edge computing* is a distributed computing paradigm that brings computation and data storage closer to the source of the data [122, 120]. Since logical network proximity is entirely characterized by low latency and low jitter, edge computing reduces communication delays as well as the size of payloads transferred between the Internet and public/private data centers.

Combining cloud computing and time-critical systems is a complex problem that requires tackling not only network-related challenges but also those related to efficient (and predictable) access to the physical platform, as well as resource management techniques that consider time-criticality [58]. Considering what has been discussed so far, we state that a truly time-critical cloud platform requires the employment of highly reliable and latency-predictable networking, computing, and storage technologies, coupled with time-aware, fault-tolerant resource management mechanisms. Until all these requirements are jointly met, no cloud provider will be able to take responsibility for establishing a proper Service Level Agreement (SLA) for stable and predictable levels of service.

This article provides a survey of the research literature on cloud solutions enabling deployment of time-critical use cases and intends to answer the following questions:

1) What are the current challenges associated with using cloud technologies for time-critical applications?
2) What cloud solutions include mechanisms for high reliability and time-predictability?

Contrary to most previous surveys [79, 58, 66, 47, 130, 9, 116], we answer these questions by investigating the entire cloud architecture software stack. We perform a broad review of papers, utilizing a breadth-first approach to cover several aspects and challenges intrinsic to cloud systems, that are critical to support time-predictable services. Figure 1 presents a typical cloud computing architecture in which the physical components are represented at the bottom of the figure. As we go from the bottom to the top, we increase the abstraction level all the way up to the applications. The management activities, responsible for the coordination among the cloud resources, are represented by the vertical box on the right. In this paper, we cover all technical aspects presented in Figure 1, and the colors used in this figure represent the way we divided the topics in this survey: Compute (green), Orchestration (orange), Storage (blue) and Communication (red).

The rest of this paper is organized as follows: Section II describes the evolution of machine virtualization techniques, showing how their performance and support for time-critical applications improved over time. In Section III, we present orchestration techniques in cloud infrastructures. We cover topics from resource management to fault tolerance and serverless computing. Section IV presents an overview of storage technologies with latency control capabilities and performance guarantees for time-critical applications. Section V is dedicated to time-critical communication techniques. The end of each section includes a discussion on the surveyed state-of-art and challenges in achieving time-predictability. Finally, Section VI draws conclusions on the current development state of a holistic cloud-based system for time-critical applications.

II. COMPUTE

This section discusses key challenges and proposed solutions for virtualization of the execution environment when executing time-critical applications. General-purpose machine virtualization mainly focuses on spatial protection and isolation (for example, the memory or files used by an application should not be accessible by others). However, for time-critical applications, we need also *temporal* isolation, i.e., the ability

of an application to respect its temporal constraints should not depend on the other applications that do not interact with it. To do this, a virtualization system has to employ *appropriate scheduling and resource allocation* to provide *low and predictable latency* [6]. These requirements are discussed with a focus on hypervisor-based Virtual Machines (VMs), containers, and unikernels, in the next sections. The main characteristics of the various papers are summarized in Table II.

A. Virtual Machines

Traditional hypervisors can provide a limited form of *temporal isolation* among hosted execution environments with their ability to partition the available physical resources, e.g., via core pinning and no over-subscription of CPU, memory and storage. Strong temporal isolation with a finer-grained allocation of resources is achievable only by using specialized solutions that combine real-time (RT) kernels and hypervisors, appropriate resource scheduling, and cache partitioning techniques, like cache coloring [88] or Intel's Cache Allocation Technology[1]. However, these are used mostly in safety-critical RT embedded systems [58]. For other domains, a few solutions approximate the concept enriching general-purpose OSes and hypervisors with various mechanisms.

For example, when using KVM on Linux with more VMs sharing the same physical core(s), the bandwidth control[2] mechanism allows for controlling the maximum fraction of CPU time consumed by the virtual CPUs (vCPUs) of each VMs, but there is no guarantee on the minimum fraction of received CPU time, or on the timing of the allocation. As a result, this mechanism cannot guarantee the respect of tight temporal constraints for the virtualized applications. In [83], the Xen credit (proportional-share) scheduler has been improved for soft RT workloads, and used for reducing the response times of a SIP server, albeit it is unable to provide formal temporal guarantees. In RT-Xen [153], a theoretical-founded RT scheduler has been added to Xen, allowing to perform a formal schedulability analysis of the guests' RT tasks based on the Compositional Scheduling Framework (CSF) [123, 124]. This algorithm, based on the Earliest Deadline First (EDF) and deferrable server algorithms, is now integrated into the mainline Xen under the name of Real-Time-Deferrable-Scheduler (RTDS). It has also been integrated in OpenStack (see Section III-A). This is a step forward to support RT applications, but a mechanism to dimension the scheduling parameters is still missing [44].

RT scheduling of a VM vCPUs can be used also with type II hypervisors such as Kernel Virtual Machine (KVM), using a RT scheduler in the host kernel. For example, the IRMOS RT scheduler [46] implements the Constant Bandwidth Server (CBS) algorithm [3], based on EDF, for groups of threads. A related old line of works include *real-time service-oriented architectures*, for example as realized in the RI-MACS [42] and Llama [109] projects, where similar RT CPU schedulers were used to serve distributed workloads in a factory automation context. However, most of these older works belong to the pre-Cloud era, so they were not considering several Cloud-specific aspects, like virtualization, containers, orchestration, load-balancing, or even multi-processor architectures.

As of today, the CBS algorithm is available in the mainline Linux kernel as the new `SCHED_DEADLINE` RT scheduling policy[3]. This allows for using hierarchical RT scheduling for the KVM vCPU threads without patching the host kernel [2].

Besides RT CPU schedulers for VMs, other mechanisms to provide low latency have also been investigated. In [4], the latency introduced by Xen and KVM is evaluated, showing that KVM can provide RT performance comparable with the one experienced by Real-Time Operating Systems (RTOSs) running on bare metal, while Xen needs some fixes [5]. Some other hypervisors, such as ACRN [85] and Jailhouse [117], have been developed from scratch focusing on latency reduction. In many cases, this goal is achieved by dedicating one entire physical CPU core to each vCPU, avoiding any kind of scheduling. This approach is also used by separation kernels [148] or partitioning hypervisors which are commonly used in embedded systems (see for example Bao [94], Xtratum [41], or the SEL4 microkernel [74]). However, these hypervisors are mostly used in embedded hard real-time use cases or simple edge real-time computing scenarios [108]. They support a limited number of hardware architectures typically employeed in specific application domains, like automotive, for which the software stack often needs to be developed according to expensive certification processes. Overall, these are more focused on hosting simple RT controllers, rather than supporting a full-featured Linux OS, for example, thus they're not usable in general cloud computing infrastructures.

Some works aim at improving the start-up time of VMs for elastic and serverless workloads. For example, the Fire-Cracker [10] "micro-VM" is a hypervisor still based on the KVM kernel module but dropping its dependency on Quick EMUlator (QEMU). This results in limited features compared to traditional Virtual Machine Monitors (VMMs), but the solution is ideal for services such as AWS Lambda that need to start many containers/VMs in a short time, even with some loss in disk throughput and I/O latency [10].

B. Containers

Containers can provide better performance than VMs, with the ability to even reach bare-metal levels, but are traditionally considered less secure, due to the need for sharing the host machine kernel. They also suffer of a weaker form of isolation: for example, in [87], the lack of I/O performance isolation on disk access for docker containers is highlighted, and an optimization of the I/O concurrency level is proposed to control the performance of each container.

Similarly to what stated above on VMs, containers allow for a very limited form of temporal isolation by employing partitioning of the physical resources, and limited CPU scheduling abilities. The latter are usually provided through the bandwidth control mechanism mentioned above, supported by most of the current container managers [150]. For example, tuning

[1]https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-cache-allocation-technology.html

[2]https://docs.kernel.org/scheduler/sched-bwc.html

[3]https://www.kernel.org/doc/Documentation/scheduler/sched-deadline.txt.

TABLE II
SUMMARY OF THE KEY CHARACTERISTICS OF THE REVIEWED PAPERS IN THE "COMPUTE" DOMAIN.

Paper(s)	Virtualization Mech.	Main Contributions	Timing Properties & Guarantees	Benchmarks
[109]	-	RT Scheduling of SOA and Java tasks, no SMP	Temporal Isolation w/CBS	Synth, Real
[42]	-	RT Scheduling of web services, no SMP	Temporal Isolation w/CBS	Synth
[88]	L4 $\mu-$kernel	Cache Coloring	Temporal Isolation	Synth
[83]	Xen	Modified Credit Scheduler	Reduced Latency	Synth
[153]	Xen	Multiple fixed-priority schedulers	Schedulability Guarantees w/CSF	Synth
[67]	KVM/HVM, Docker	Experimental Comparison	-	Synth
[46]	KVM	RT Scheduling of VMs	Temporal Isolation w/CBS	Synth
[2]	KVM	SCHED_DEADL. on KVM vCPUs	Temporal Isolation w/HCBS, Analysis w/CSF	Synth
[4, 5]	KVM, Xen	Experimental comparison	Reduced latency	Synth
[10]	Firecracker	KVM w/o QEmu	Reduced latency and boot time	Synth, Real
[87]	Docker	Perf. Isolation for containerized storage services	Reduced Latency, Increased Throughput	Synth
[150]	Docker	Flexible Deferrable Server	CPU Reclaiming for Non-RT	Synth
[1, 43]	LXC, OpenStack	Hierarchical RT Scheduling	Temporal Isolation w/CSF	Synth
[93]	Docker, K8S	RANs RT Cloud with PREEMPT_RT and DPDK	Reduced Latency	Real
[37]	Docker	FP Scheduling w/RTAI	Reduced Latency, No temporal Isolation	-
[39, 38]	Docker	FP/EDF Scheduling w/RTAI	Temporal Isolation	Synth
[136]	Xenomai, Docker	Xenomai w/PREEMPT_RT Linux containers	Reduced Latency	Synth
[24]	Xenomai	RT scheduling of containers with RT networking	Temporal Isolation w/co-kernel	Synth
[78]	Unikernels	UKs for NFV	Better Latency/Throughput	Synth
[98]	Unikernels	UKs for Serverless Comp.	Reduced Latency	Synth
[34]	Unikernels	Deferrable Server for UKs	Timing Guarantees via CSF	Synth

Acronyms: Real-Time (RT), Radio Access Network (RAN), Network Function Virtualization (NFV), Fixed Priority (FP), Earliest Deadline First (EDF), Constant Bandwidth Server (CBS), Kubernetes (K8S), Symmetric multiprocessing (SMP), Unikernels (UKs), Synthetic/Realistic workload (Synth/Real).

the Linux CPU scheduler for improving the responsiveness of real-time packet processing in Cloud-RAN scenarios has been proposed in [106]. However, to provide real-time guarantees to applications, a formal schedulability analysis is needed, like the CSF [123, 124] mentioned in Section II-A. Indeed, implementations of CSF have been realized not only for hypervisors [153, 2], but also for containers. For example, the Hierarchical CBS (HCBS) has been proposed in [1], an extension of the `SCHED_DEADLINE` CPU scheduler in the Linux kernel. HCBS allows to attach groups of tasks to a multi-processor CBS, where individual tasks are scheduled using the standard priority-based RT scheduler within the time-slices allocated to the group. These techniques can also be used to provide guarantees to complex applications, with interacting tasks and more flexible performance guarantees based on probabilistic analysis [43].

Other works investigated how to reduce the latency of containerized applications (see [130] for a survey). This can be done by using a preemptible host kernel or by using a co-kernel approach. The authors in [93] evaluate the latency introduced by Docker containers with various kinds of kernel (standard, low latency and fully preemptible — Preempt-RT), showing the advantages of using Preempt-RT and DPDK for low-latency networking. In RT-CASES [37], the RTAI co-kernel was used to provide containers with low scheduling latency. However, the first proposed mechanism supported only fixed-priority scheduling, so there was no temporal protection among RT threads. The approach was extended in [39], adding a monitoring component to cope with RT threads trying to overrun their known WCET, still using the fixed-priority scheduler available in RTAI. However, monitoring was based on a sampling approach, which implies either low precision or high overheads, with authors suggesting dedicating a CPU just for the monitoring thread. Finally, EDF scheduling with hard CBS was added in [38], modifying the RTAI scheduler.

Another work proposing a co-kernel can be found in [136], where low latency in containers is achieved by combining Xenomai and Linux with a Preempt-RT kernel. In [67], Preempt-RT, Xenomai and some instances of the AWS EC2 cloud using the HVM hypervisor are compared in terms of latency measured with cyclictest. Authors claim that some HVM-based instances can be suitable for migrating time-critical industrial control applications from bare metal to IaaS services, using a Preempt-RT Linux kernel, but they neglect the implied impact of network delays in their evaluation. Another interesting work can be found in [24], where Xenomai is extended with a hierarchical reservation-based CPU scheduler, called SCHED_DS, to obtain real-time scheduling of containers. Basically, thread groups from different containers are scheduled using a deferrable server algorithm, then within each container threads are scheduled by priority. The solution relies on the Xenomai RTnet [73] module to add also TDMA-based real-time networking guarantees on Ethernet networks.

C. Unikernels

VMs are often preferred to containers thanks to their improved isolation. However, they bring higher overheads due to the higher complexity of the overall software stack on the node, resulting in higher latency/execution times, boot times, and memory footprint. Unikernels aim to mitigate these drawbacks, reducing the complexity of a VM software stack. Indeed, a unikernel is a library Operating System (OS) directly linked with the target application, and deployed in a VM; see [35] for a survey with detailed information on unikernels and containerization solutions for edge computing.

Various papers investigated on performance improvements and overhead reductions for unikernels. In [78], unikernels are

proposed to get the performance of containers with the security of VMs, with experiments performed with IncludeOS[1]. In [98], a serverless platform is proposed based on unikernels (SOLO5), comparing the achieved performance with the one of Apache OpenWhisk[2], based on Docker containers.

Some studies [95] show that unikernels can even provide lower latency than Docker-based containers in some situations. However, these performance improvements depend on the unikernel used for the experiments: a performance comparison [28] carried out using the *rumprun*[3] unikernel shows a decrease in memory and CPU usage compared to KVM, but also increased latency and lower throughput.

Finally, the work in [96] presents a more systematic benchmarking of the various technologies that can implement a container (hypervisors, kernel virtualization, micro-VMs, intercepting system calls, unikernels, ...) showing that the combination of micro-VMs and unikernels allows to greatly reduce the overhead introduced by hypervisor-based virtualization.

Unfortunately, most of the research on unikernels did not consider the requirements of time-critical applications. The only notable exception is the work presented in [34], which shows how unikernels allow for better timing guarantees for single-threaded applications, using an optimized version of the traditional CSF analysis, which is anyway quite pessimistic.

D. Discussion

Summing up, in the compute area, there has been significant academic research on virtualization techniques to provide real-time CPU scheduling. Some of these works produced mechanisms that have been integrated into mainline OS kernels and hypervisors. As a result, two of the most commonly used open-source hypervisors (KVM and Xen) are suitable for being used in time-critical use cases, providing isolation, low latency, and predictable real-time CPU scheduling. On the other hand, these features are available for containerized time-critical workloads only applying out-of-tree patches. Various works focused on the advantages of using unikernels, in terms of boot-time reduction for example. However, this technology does not seem mature enough to be used in production environments and the impact of unikernels on time-critical applications has not been fully evaluated. There are also numerous other operating system and hypervisor solutions in the embedded and real-time systems domain, addressing real-time CPU scheduling, including also commercial products such as VxWorks[4], Helix[5], Erika Enterprise[6], PikeOS[7] or others, mostly providing certified solutions for specific application domains, but often lacking portability on general-purpose commodity hardware, as needed in cloud computing. Even though the basic technologies to provide predictable real-time CPU scheduling already exist, there is no mechanism to properly dimension the scheduling parameters based on the application requirements. In addition, CPU scheduling has not been integrated with other resources such as network and storage in order to support time-critical applications that are not purely CPU-intensive.

[1]More information is available at: https://www.includeos.org/.
[2]More information is available at: https://openwhisk.apache.org/.
[3]For more information, see: https://github.com/rumpkernel/rumprun.
[4]More information at: https://www.windriver.com/products/vxworks.
[5]More information at: https://www.windriver.com/products/helix.
[6]More information at: https://www.erika-enterprise.com/.
[7]More information at: https://www.sysgo.com/pikeos.

III. Orchestration

Performance-aware cloud orchestration has been studied at large, encompassing several challenges related to time-criticality, like: efficient resource scheduling and allocation with isolation among virtualized tasks; fault tolerance; optimized serverless architectures; and, RT data streaming. For instance, the Kubernetes scheduler has been enhanced by leveraging system-level metrics [145] to reduce interference, or using spare backups and ping-based latency-awareness [142] for reliability and low-latency. However, for time-critical workloads, we need proper mechanisms ensuring the end-to-end latency is not disrupted by transient problems nor faults. Moreover, most of the papers in this area consider tasks with relatively long deadlines, as found in big-data or scientific workloads, but some papers covering low latency in the micro/milli-second range exist as well. Table III summarizes key aspects of the works on orchestration discussed below.

A. Task Resource Scheduling & Admission

To provide stable performance, cloud services use extensively load balancers to spread the load across a pool of instances, and horizontal elasticity to scale the pool so to match target performance levels. A number of works exist [97, 31] surveying key approaches to load-balancing and elasticity in cloud computing. However, most of the elasticity approaches are merely *reactive*, i.e., they correct the system behavior once it deviates from the desired state, meaning that they do not include effective mechanisms for *temporal isolation*.

RT workloads may be hosted in existing cloud management solutions by leveraging on resource partitioning to obtain similar performance as with a dedicated physical machine (as supported by AWS or GCP bare-metal instances, or the Ironic[8] service in OpenStack, for example). This gets rid of temporal interferences among instances, but it also cancels several advantages in adopting cloud infrastructures. When switching to regular instances that share physical servers, many instance types offered by cloud providers offer a 1:1 mapping between virtual cores with physical ones. This is also supported in OpenStack[9], which also allows to configure a pre-emptable real-time kernel, tune the Dynamic Voltage and Frequency Scaling (DVFS) and disable deep idle states on the host to reduce response latencies. However, none of these solutions support the use of RT scheduling policies (i.e., CBS) to ensure proper temporal isolation among virtual cores when sharing physical cores.

To overcome these limitations, the IRMOS project [46] proposed a cloud management solution for interactive real-time and multimedia distributed applications. It included the Intelligent Service-Oriented Networking Infrastructure (ISONI) [45], enabling deployment of complex distributed real-time software

[8]More information at: https://ironicbaremetal.org
[9]See: https://docs.openstack.org/nova/wallaby/admin/real-time.html

TABLE III
SUMMARY OF THE KEY CHARACTERISTICS OF THE REVIEWED PAPERS IN THE "ORCHESTRATION" DOMAIN.

Domain	Papers	Main Contribution	Timing Property & Guarantees	Benchmark
Resource Scheduling & Admission	[43]	Control the interferences of co-located real-time services	Temporal isolation	Synth, Real
	[152]	Co-hosting of RT and non-RT VMs on OpenStack	Schedulability experimental evaluation	Synth
	[54, 129, 128]	RT scheduling on K8S	Temporal isolation w/HCBS	Synth
	[91]	RT scheduling on K8S + ROS	Temporal isolation w/SCHED_DEADLINE	Synth, Real
	[25]	Criticality-aware scheduling in K8S	Temporal isolation	Synth
	[51]	Time-slice control scheme for VMs	Reduced exec. time	Real
	[33]	Energy-efficient RT scheduling in Cloud	High deadline guarantee ratio	Synth, Real
	[65]	Response time analysis for aperiodic time-critical services	-	Synth, Real
Fault Tolerance	[75]	Low latency extension of Raft	Reduced tail latency	Synth, Real
	[59]	High-performance consensus protocol	Reduced latency	Synth, Real
	[11]	RDMA-based consensus protocol	Reduced repl. latency	Synth
Serverless	[103]	SLO-aware OpenFaaS extension	Function invocation rate guarantees	Synth
	[135, 134]	partitioned-EDF scheduling for RT functions	Schedulability experimental evaluation	Synth
Real-time Stream Processing	[100]	Timely dataflow system	Reduced latency, increased throughput	Synth, Real
	[56]	Congestion-aware scheduler in Apache Storm	Reduced latency, increased throughput	Real
	[156]	Locality-aware resource allocation, co-flow scheduling	Reduced latency, increased throughput	Real
	[102]	RT Apache Storm	Temporal Isolation	Real

Acronyms: Real-Time Virtual Machine (RT VM), Best-Effort Virtual Machine (non-RT VM), Synthetic workload (Synth), Realistic workload (Real), Kubernetes (K8S), Remote Direct Memory Access (RDMA).

components in the form of Virtual Service Networks (VSNs), i.e., Directed Acyclic Graphs (DAGs) of VMs, with attached precise end-to-end temporal constraints, expressed in either deterministic or probabilistic terms. These constraints could be respected thanks to the use of the above mentioned IRMOS real-time CPU scheduler based on the CBS, standard packet schedulers supporting QoS guarantees, and proper storage management, as well as optimal VM placement [76].

More recently [43], a modified Nova component in OpenStack has been proposed: on the compute side, time-critical VMs are deployed by setting the required RT scheduling parameters on the underlying HCBS RT scheduler; on the controller side, it embeds a placement logic that is aware of the RT computational bandwidth allocated to each time-critical instance. The solution is validated through extensive experimentation, where multiple co-located benchmarks from the domains of Network Functions Virtualization (NFV) and IP Multimedia Subsystem (IMS) are deployed in a real test bed. The results show that the proposed mechanism ensures temporal isolation of individual containerized activities. Similarly, RT-OpenStack [152] is a modified version of OpenStack for co-hosting time-critical and regular VMs, by: (1) integration of the RT-Xen hypervisor within OpenStack through a RT resource interface; (2) a RT VM scheduler to enable regular VMs to share hosts with RT VMs without temporal interference; and (3) a VM-to-host mapping strategy to provision RT performance to RT VMs while allowing effective resource sharing with regular VMs. In practice, RT-OpenStack performs RT admission control in the OpenStack compute scheduler. Then, RT VMs are scheduled using global EDF.

Kubernetes has also been modified with real-time scheduling features, to host time-critical containers. RT-Kube [91] integrates Kubernetes directly with the SCHED_DEADLINE scheduler available in the mainline Linux kernel to support scheduling of RT ROS tasks and management of the CPU bandwidth. The authors provide experimental results based on synthetic benchmarks, and a real robot control use-case in a pick-and-place scenario. However, the proposal relies merely on the default configuration using Global EDF and an admission test which is only necessary, not sufficient to guarantee schedulability of RT tasks on multiprocessors. Moreover, the paper does not discuss real-time networking, nor how they reconciled SCHED_DEADLINE, which deal with single-threaded processes only, and the typically multi-threaded architecture of ROS applications. RT-Kubernetes [54] addresses this issue by using the HCBS extension to SCHED_DEADLINE, which allows setting a different runtime on each core/CPU to avoid resource waste on many-core hosts. This contribution is also coupled with an admission test for finding proper nodes according to the available real-time CPU bandwidth. A quite similar approach can be found in [129, 128], which is also based on the same HCBS real-time scheduler. Compared to RT-Kubernetes, the authors implemented a feedback loop that continuously monitor the resource usage and QoS experienced by the containerized workload through a set of metrics (e.g., deadline misses, lateness, response time), so to consider this information in future scheduling decisions. Another paper exploring similar aspects with real-time task group scheduling can be found in [25]: the main differences lie in details of the admission test, placement strategy, and monitoring mechanisms.

Adaptive Time-slice Control (ATC) [51] is a solution to detect communication and computation phases in applications based on lock latency and cache misses. This information is used to shorten time-slices during communication phases and prolong them during computation phases. The authors focus on multithreaded and distributed applications including multiple interacting VMs. The method chooses longer time slices during compute phases to keep the cache filled. The evaluation is done for applications with clear communication and compute phases like distributed compilation.

In [33], Proactive and Reactive Scheduling (PRS) is pro-

posed for VMs with uncertain execution times and aperiodic real-time tasks, considering energy efficiency i.e., minimizing the active hosts while guaranteeing timing requirements. The scheduler is based on the task laxity, accounting for both the task durations and deadlines. In [65], cloud services with probabilistic latency guarantees are presented, with a stochastic response time analysis for aperiodic services following a Poisson arrival process on computing platforms that schedule time-critical services as deferrable servers.

B. Fault Tolerance

We can find both proactive and reactive approaches to fault tolerance in cloud computing [66, 72]. Among the former ones, there are preemptive migration and rejuvenation, which reduce the likelihood of a fault occurring by moving the task away from the potentially faulty system in advance, or by making sure to repair the system before the fault occurs. Among the reactive methods, checkpointing and replication are common, which enable the possibility to replay tasks at another instance after a fault occurred. For real-time cloud applications, most of the reactive approaches are based on replication due to the relatively low latency recovery time. On the other hand, checkpointing requires at least a couple of seconds in downtime during state snapshot for common containerized application state sizes. Moreover, the cloud management and orchestrator itself may have a variety of failures, as highlighted in [27], where a Kubernetes fault injection framework is proposed, based on several reported observations from real deployments. The same authors provided also an interesting analysis [26] of the time needed to handle and repair detected faulty pods in Kubernetes. However, only a few papers deal with fault tolerance for cloud services with latency requirements in the millisecond range. This is interesting for some of the more demanding use cases like tight automation control systems. These are commonly handled today by dedicated embedded hardware, but they are expected to see much increased compute requirements, so that offloading to a cloud system will be beneficial [126, 157].

Both scalability and fault-tolerance can be handled with the same design principle of replication. However, for elastic services, too many replicas improve fault-tolerance at the cost of hurting the performance. This is due to the fact that consensus overhead grows with the number of nodes. In [75], it is explored if adding nodes can both increase performance and fault tolerance for stateful data-center applications that require fault-tolerance, low latency, and scalability. They present HovercRaft, a load balanced Remote Procedure Call (RPC) protocol extended with a Raft derivative that separates the ordering and replication of requests. The replies are sent from any of the replicas, reducing the work for the leader replica and increasing scalability. The load balancing policies reduce the tail latency but provide no deadline guarantees.

Another interesting work can be found in [114], where a strategy is presented to reduce latency variations when using load balancing and request redundancy, i.e., *hedging*. Measurements of microsecond-level variations in request latency show that excessive redundancy may lead to congestion and increased latency variations, whilst redundancy permits using the earliest response, achieving a more consistent latency.

Nezha [59] is a high-performance and deployable consensus protocol that exploits accurate software clock synchronization. It does not require special hardware or physical network access, making it easily deployable in virtualized environments. Instead, it uses a new primitive to send requests to several replicas by multicast which orders client-to-replica requests by deadlines. The deadlines are specified in the synchronized wall clock time. Performance is evaluated for two applications, Redis and a prototype financial exchange CloudEx, which show fault tolerance is achieved with only a modest performance degradation.

Another low latency fault tolerant replication scheme using leader consensus is based on Remote Direct Memory Access (RDMA) [11], introducing a new state machine replication (SMR) protocol, called Mu, that carefully leverages RDMA to lower the fault detection time. Mu uses a conceptually different method based on a pull-score mechanism over RDMA. The leader increments a heartbeat counter in its local memory, while other replicas use RDMA to periodically read such counter. A badness score is calculated based on the number of successive reads that have returned the same value. Replicas declare a failure if the score is above a badness threshold, corresponding to causing a timeout. Unlike traditional heartbeat signals, this method can use an aggressively small timeout without false positives because network delays slow down the reads rather than the heartbeats. This way, Mu detects failures usually within 600 microseconds. When a failure occurs, this scheme is shown to take less than a millisecond to recover, whereas HovercRaft takes 10 milliseconds [75].

Fault detection is a related research field dealing with observability challenges both at the infrastructure and application levels. In time-critical services, faults manifest not only as conventional infrastructural problems (i.e., network issues or hardware failures) but also as deadline misses. However, only a few works deal with low latency fault detection for real-time cloud systems. For example, in [7, 17, 16], a low-latency detection system is proposed, alongside a performance model that makes use of timely fault detection to evaluate the feasibility of time-constrained, cloud-native applications.

C. Serverless Computing

The serverless paradigm provides event-driven services that relieve developers from the burden of server provisioning and network infrastructure management. In Function-as-a-Service (FaaS), developers deploy functions i.e., isolated and stateless components processing inputs to obtain outputs, using containers in a cloud environment (e.g., a Kubernetes cluster). These are created, scaled and shut down on-demand, according to the instantaneous workload. This structure is very useful in event-driven non-critical IoT but tends to exhibit large latency and jitter. One of the issues with FaaS is the “cold” start-up time, incurred when an instance is not available yet, on a function invocation. Another criticality is due to the fact that often applications offload several parts of their computations to a chain of function invocations. To create stateful applications, cloud storage services are also needed.

Existing serverless platforms do not support real-time applications in the deterministic sense. An extension to OpenFaaS has been proposed [103] for Service Level Objective (SLO) guarantees in terms of functions' invocation rate. More specifically, an admission controller accepts the deployment of functions based on the system state and requested invocation rate guarantees, and a predictive manager provisions the underlying containers accordingly.

RT-FaaS has been proposed [135, 134] for services that can execute a mix of real-time and non real-time functions, providing the former with response time guarantees. A heuristic spatial allocator is used to minimize the processors allocated to RT tasks. The framework assumes real-time communications ensuring known timing of data transfers in remote service invocations, and knowledge of the worst-case processing times. On the hosts that execute the functions, a per-function EDF scheduler is used to queue and serve the requests.

D. Real-time Stream Processing

Many time-critical applications require low latency interactive access to results of data stream processing pipelines, such as real-time analytics of data coming from IoT or wearable devices. In this context, the potential value of the extracted information rapidly decreases over time, hence the need for stream processing platforms designed for high responsiveness. One of the first works in this context is Naiad [100], a general-purpose low-level programming abstraction for executing distributed data-parallel, cyclic dataflow programs with mixed high throughput and low latency requirements. It implements a novel computational model which enriches dataflow computations with logical timestamps to track the global progress of the computation, together with an efficient notification-based coordination protocol that allows for several communication policies, such as prioritizing the delivery of notifications with the earliest timestamp to reduce end-to-end latency.

Most of the existing processing frameworks have been designed under the assumption of an abundance of memory and powerful computing resources, resulting in inefficient resource usage and unpredictable I/O costs. In the context of time-criticality for stream processing engines, this is tackled by focusing on resource-constrained edge computing scenarios, and on the idea of processing data streams "close to the data source". EdgeWise [56] modifies Apache Storm[1] to improve both throughput and latency on Edge-based Storm deployments. It introduces a fixed-size worker pool execution model, replacing the one-worker per-operation architecture of Storm, and a congestion-aware user-level scheduler that ensures I/O queue lengths do not exceed available memory by assigning a ready thread to the operation with the most pending data. Both changes reduce the intrinsic non deterministic nature of general-purpose OS scheduling, making stream processing more amenable for memory-constrained edge environments. Amnis [156] is a novel stream query processing engine that extends Apache Storm to tackle the challenges in edge computing scenarios. Amnis optimizes the throughput and end-to-end latency of data streams by: employing a data locality-aware resource allocator for query operators; adopting a load-aware scheduler that considers the dynamically varying load conditions and resource requirements of each operator; and considering the dependency between data flows to be transferred through the network so that the volume of data can be controlled with a rate limiter. Finally, Storm-RTS [102] replaces the default worker-based execution model of Apache Storm with a rate-based abstract machine, building a stream processing workflow as a chain of rate-guaranteed function invocations. This allows for flexible resource reconfigurations achieving predictable performance.

[1] https://storm.apache.org/

E. Discussion

In the orchestration research area, we noticed a great focus on best-effort cloud services or long duration workloads, where the goal is to ensure that the majority of users receive a satisfactory service. For example, it is probabilistically harmless to have temporal service disruptions due to a noisy neighbourhood problem, or during a scale-out operation or the replacement of a faulty instance. However, this is in contrast with the traditional real-time deterministic view of per-request latency guarantees, as needed by time-critical applications. Many researchers dealt with the potential of horizontal elasticity, with a shortage of efforts in enriching cloud management solutions with key features, such as: integration with strong temporal isolation features among instances sharing physical resources, as found at the hypervisor or OS levels in compute nodes (see Section II), with a fine per-request granularity control on the end-to-end latency, and low latency fault detection and recovery. Fault tolerance is a standard concept in cloud and distributed computing, but it is mainly implemented without worst-case latency guarantees. Only a few approaches tried to integrate cloud technologies with worst-case guarantees despite faults, as needed by time-critical applications. This confirms the need for further investigations in this direction.

IV. Storage

Modern cloud-native data-driven applications with time-critical requirements, such as multimedia streaming and cloud gaming platforms, need specialized storage services with latency control capabilities and performance guarantees. To this end, cloud providers must employ optimized software stacks for local storage nodes, as well as ad hoc database architectures equipped with specialized task and resource schedulers. Most of the literature does not employ solid real-time methods, focusing instead on adaptive latency control mechanisms for the latency-sensitive use case. In the following, the existing literature is summarized, with a focus on: i) local storage optimizations that employ low-level mechanisms to control latency, such as cutting-edge storage drives and filesystems optimized for virtualized environments; and, ii) the goals and challenges of traditional real-time databases, presenting the current solutions for the control and/or reduction of latency. A summary of the key characteristics of the papers reviewed in this section is provided in Table IV.

TABLE IV
SUMMARY OF THE KEY CHARACTERISTICS OF THE REVIEWED PAPERS IN THE "STORAGE" DOMAIN.

Paper(s)	Storage Context	Main Contributions	Timing Properties & Guarantees	Benchmarks
[147]	I/O scheduler	Reservation-based exclusive disk access	High throughput with fairness guarantees	Synth, Real
[29, 30]	SSD Subsystem	Storage interface exposing SSD internals to the host	I/O predictability and isolation	Synth
[90]	SSD Opt	Leverage SSD internals to reduce in-device R/W contention	Reduced tail latency	Synth
[82]	SSD Opt	Hybrid polling scheme for Ultra-low latency SSD	Reduced CPU utilization and I/O latency	Synth
[151]	I/O Stack Opt	Layer-aware I/O stack for containers to reduce host-level resource contention	Reduced Latency	Synth
[132]	Overlay FS	Block-level accessibility for containerized storage systems	Reduced average latency	Synth, Real
[52]	Container image	File sharing between images and on-demand file retrieval	Reduced storage space usage	Synth
[69]	RTDB	QoS-aware feedback control for bursty workloads	Bounded response times	Real
[70]	RTDB	QoS-aware feedback control for embedded storage	Minimize real-time transaction tardiness	Synth
[154, 155]	NoSQL	QoS and QoD aware task scheduling	Minimize requirement violations	-
[101]	In-memory NoSQL	Transparent, locality-optimizing data migration for ultra-low data access	Reduced latency	Synth
[15]	NoSQL	Priority-based performance differentiation	Reduced/stable latency for high-priority req.	Synth, Real
[149]	NoSQL	Multitiered data store with adaption to workload changes	Cost-vs-performance/latency tradeoff	Synth
[112, 49]	NoSQL	Fully-managed cloud service with differentiated performance	Guaranteed throughput and low latency	Real
[127]	NoSQL	Preserving ACID transactions keeping linear scalability	Reduced average latency	Synth
[63]	NoSQL	Fog and Context aware replica placement and consistency	Reduced latency	Synth
[118]	NoSQL	Autonomic rate limiter for latency-critical edge	Reduced latency	Synth
[105]	NoSQL	Mobility and Access pattern aware data placement	Reduced tail latency	Synth, Real

Acronyms: Real-Time Database (RTDB), NoSQL Database (NoSQL), File system (FS), Optimization (Opt), Synthetic/, Realistic workload (Synth/Real).

A. Local Storage Optimizations

Before the introduction of Solid State Drives (SSDs), a typical source of unpredictability was the high seek latency of traditional rotational disks, a problem tackled in two ways: using in-memory storage only, as proposed in [101], or by employing special disk access scheduling techniques coupled with pessimistic analysis, such as the Budget Fair Queuing (BFQ) scheduler [147], which combines disk idling with timestamp-based scheduling to preserve both guarantees and high throughput for synchronous requests. BFQ reserves a fraction of the disk throughput and ensures exclusive disk access for a certain amount of budget time every period for each application. Although SSDs eliminated the high seek latency problem, a traditional block I/O SSD suffers from unpredictable performance due to the often non-disclosed sector allocation, prefetch, and caching logic within the drive controllers, as well as suboptimal resource utilization. Hence the introduction of open-channel SSDs for fine-grained, application-specific management of the data placement strategy as well as of the physical I/O scheduling. LightNVM [29] is a novel I/O subsystem within the Linux Kernel that exposes the parallelism capabilities and storage media characteristics of open-channel SSDs to the host OS. LightNVM provides an interface where application-specific abstractions can be implemented. A recent work that makes use of this interface is RAIL [90], a novel SSD management technique to reduce read tail latency in the presence of write operations. The authors state that since write buffer flushes can contend with reads on flash devices and bring long latency tails, RAIL redirects user and garbage collection writes towards independent physical flash chips, thus exploiting the internal parallelism of NVME SSDs in order to eliminate the possibility of reads being stalled by any high latency operation (read-after-write). RAIL effectively enforces predictable performance for Flash accesses, achieving 7× lower tail read latency than conventional SSD management techniques. Open-channel SSDs have been replaced by Zoned Namespace (ZNS) SSDs [30], a standardized, media-agnostic interface among different SSD vendor implementations.

The recent introduction of ultra-low latency SSDs naturally shifted the direction of I/O performance improvements towards the I/O stack, which now accounts for a large fraction of the application-perceived I/O latency. An adaptive hybrid polling scheme has been proposed [82] that combines polling and interrupts. The proposal makes optimal sleep time decisions to maximize the I/O performance and minimize the CPU cycles used for polling. An optimized layer-aware I/O stack has been proposed in [151] to reduce file search overheads for containers and alleviate resource contention in the native file system. This is done in two steps: i) by employing a modified virtual file system that locates files based on layer information to enable simultaneous Copy-on-Write (CoW) operations, thus reducing file open and lock contention; and ii) by replacing the journal service of the native file system with multiple, isolated micro-journals per container, thus reducing resource conflicts caused by shared resources. Experimental results show improved latency and throughput for a set of containerized data-intensive applications.

Baoverlay [132] is a lightweight overlay filesystem that enables block-level access for improved write performance in container-based storage systems. It logically decomposes an overlay container image file into equally-sized blocks so that a data update originated by the container does not involve copying the whole file, thus reducing the write latency overhead of a traditional CoW mechanism, which could incur into long "read-and-then-write" I/O operations.

Gear [52] is a container image format for efficient image storage and deployment. It introduces an index structure decoupled from the regular image files, which allows for on-demand retrieval of necessary files only and file sharing between different images while keeping full compatibility with Docker. The proposal effectively saves up to 54% storage space in the registry but also accelerates container deployment under scenarios with limited bandwidth.

B. Real-time Databases

Unlike a traditional database management system, a real-time one [71] employs several optimizations to minimize the number of missed deadlines by reducing to the minimum the unpredictability of transaction execution. For instance, conventional resource (i.e. CPU and I/O) scheduling algorithms are designed to ensure fairness between tasks, whereas a real-time one schedules the next transaction as a function of its criticality and the tightness of its deadline. Similarly, the conventional concurrency control protocols, used to ensure consistency between concurrent transactions, may lead to priority inversion or deadlocks. Hence the use of lock-free algorithms [158] integrating transaction priorities into the conflict resolution logic, such as the one proposed by Lindström [89]. Another challenge is to completely eliminate I/O accesses, due to the unpredictability of disk access delays, by putting data directly in memory and employing recovery protocols [92] in case of hardware failure. In the context of classic real-time database systems, there is a multitude of feedback real-time scheduling and utilization control algorithms, such as Chronos [69], which dynamically controls the desired service delay through admission control of incoming transactions in proportion to the response time error, effectively managing the amount of backlog in the database.

Similarly, QeDB [70] employs a feedback control scheme for data-intensive, real-time applications hosted on embedded systems with memory constraints. QeDB simultaneously manages both I/O and CPU resources to guarantee the desired trade-off between timeliness of transactions and data freshness. By monitoring the I/O and CPU tardiness, QeDB computes the buffer hit ratio adjustment to allocate the proper amount of resources in a robust and controlled manner.

The advent of cloud computing raised interest in cloud-hosted databases for modern large web-based workloads with time-critical requirements. This shift towards highly scalable infrastructures led to the shortcoming of relational databases in favor of NoSQL architectures, which sacrifice ACID transactions in favor of relaxed consistency models, to achieve higher availability and scalability [32]. Although there is little recent academic literature on real-time cloud-based data stores, there are many prototypes for latency-sensitive workloads that could be coupled with traditional real-time database principles to ensure predictable performance. AQUAS [154, 155] is a replacement scheduler for Apache Cassandra[1] that efficiently allocates replica nodes to minimize the penalties incurred in violating a set of QoS and Quality-of-Data (QoD) requirements. It estimates the execution time of operations via a simple linear regression model for writes, and by looking at the execution times of previous queries for reads. AQUAS offers various query scheduling policies, such as FIFO or EDF.

DAL [101] is an in-memory data backend with low latency and high reliability. It works both as a data store and a publish-subscribe message broker. A DAL cluster is made of a collection of distributed servers queried by clients. When a server detects that an item is mostly accessed from a specific remote client, it proposes a data move closer to such client. When the same physical machine co-hosts a client and a server instance, data items that are dominantly used by a single client will be accessed locally after a short transient.

RT-MongoDB [15] is a modification to the popular MongoDB NoSQL data store that makes use of POSIX nice levels and of a priority-based synchronization logic to offer reduced per-user or per-request response times: higher priority requests/clients are able to be served earlier than lower priority ones by preempting or starving the latter for arbitrarily long time windows. More specifically, the work exploits the MongoDB per-client thread model, as well as a concurrency control mechanism, to implement prioritization of the threads corresponding to higher-priority clients/requests.

Anna [149] is a cloud-native, distributed key-value store that dynamically adapts to workload variations based on three high-level goals specified by the administrator: average latency, cost–performance tradeoff, and fault-tolerance. Anna employs a policy engine that monitors the workloads and responds to variations that may violate the objectives through horizontal scaling, vertical data tiering (i.e. moving "hot" data to faster memory, demoting rarely used data to cold storage), and selective replication of "hot" keys onto multiple nodes.

DynamoDB [112] is a fully-managed NoSQL data store from AWS that guarantees consistent performance "at any scale". Based on the popular Dynamo [49] data store, DynamoDB allows customers to declare their desired read/write throughput for a given table, then the underlying infrastructure is configured so to meet these requirements with high probability. DynamoDB is the de-facto industry standard solution for real-time and predictable performance, high reliability and availability, and weak data consistency models.

CitrusLeaf [127] is a NoSQL data store that combines the consistency and reliability capabilities of traditional databases with the extreme scalability of self-sufficient clustered distributed architectures, such as Apache Cassandra and Dynamo. It dynamically redistributes data between replica nodes using real-time prioritization techniques to balance long-running tasks (e.g., batch queries or bookkeeping activities) and short client transactions with sub-millisecond requirements.

The trend of moving the computational resources closer to the "edge" of the network arose to address the concerns of response time requirement, battery life constraint, and bandwidth cost saving. In this context, FogStore [63] is a geo-distributed key-value store based on Apache Cassandra suitable for strongly consistent systems with low latency and fault-tolerance requirements. The work proposes a location-aware replica placement mechanism that tackles the conflicting problem of placing data replicas close to the relevant clients (for low latency), or in distant locations (for fault-tolerance).

[1] https://cassandra.apache.org/

An application-specific edge data store has been proposed [118] for distributed machine vision applications. A compute-intensive vision algorithm extracts image feature vectors and key-frames: the former have latency-critical requirements, whereas the latter are kept primarily for archival purposes. The work incorporates an autonomic rate limiter for transmitting key-frames that sacrifices their latency and accuracy to maintain latency-criticality for feature vectors. Early experimentation with a synthetic workload shows a median latency improvement of 84%.

Portkey [105] is a distributed key-value store considering the time-varying mobility and latency patterns of emerging IoT and mobile applications. It continuously monitors the client locations and data access patterns, and it dynamically devises a near-optimal data placement strategy to control the average/tail latency. With respect to existing NoSQL data stores, Portkey employs self-correcting greedy heuristics that prioritize fast (and frequent) approximate placement decisions over slow optimal placements, reducing tail request latency by 21-82%.

C. Discussion

In the storage research area, there is a lack of works investigating the use of low latency, predictable storage solutions for cloud applications. Investigations on "proper" real-time databases have been of interest only in the restricted domain of hard real-time systems. However, modern cloud-hosted time-critical applications may create an emerging need for novel solutions tied to the cloud computing context. Here, there are several research opportunities thanks to promising new storage technologies, such as ZNS SSDs, integration with predictable computing mechanisms as reviewed in Section II, and cloud orchestration solutions as seen in Section III. This leaves room for unexplored research opportunities in the development of cloud-based, data stores for time-critical applications.

V. Networking

Network infrastructures are becoming increasingly suitable for time-critical communications [110, 55, 146] thanks to 5G/6G technologies such as Ultra-Reliable and Low Latency Communication (URLLC), and Time-Sensitive Networking (TSN). URLLC can facilitate highly critical applications with very demanding requirements in terms of E2E latency (millisecond level), reliability, and availability. TSN aims to provide deterministic services over the IEEE standard 802.3 for Ethernet wired networks. This means guaranteed packet transport with low and bounded latency, low packet delay variation, and low packet loss. Moreover, the recent advent of NFV [50], where physical appliances sized for peak hour operations are being replaced by Virtual Network Functions (VNFs), has led network providers to increasingly adopt principles and solutions typical of cloud computing, albeit within a private cloud context. VNFs are software components designed with elasticity and scaling abilities typical of cloud services [133], and deployed on general-purpose data centers of a network operator, typically recurring to cloud platforms.

Time-critical use cases need bounded, low latency, and often deterministic E2E communications with high reliability and availability. They require a networking infrastructure meeting requirements that are different from normal best-effort, broadband services, namely each packet must meet the timing requirement, otherwise the service will not work properly. In the following, an overview of research works dealing with predictable and reliable networking is made, focusing on: solutions ensuring deterministic traffic scheduling in wired communication networks, along with the challenges for supporting it in wireless networks as well; real-time capabilities of the communication stack, encompassing both OS-level and hypervisor-level predictable packet processing, as needed in virtualized/containerized workloads to meet their timing constraints; and, real-time packet processing capabilities of the virtual switching components, so crucial for the E2E performance of distributed cloud workloads. The main aspects of the works reviewed below are summarized in Table V.

TABLE V
Characteristics of different communication solutions.

Ref.	Solutions	Latency[1]	High through.	Isol.[2]	High avail.[3]
[141]	TSN IEEE 802.1	Predictable	✓	✓	✓
[57, 119]	TSN + DPDK + K8S	Predictable	✓	✓	
[137]	5G TSN/TSC	Predictable	✓	✓	✓
[131]	5G URLLC	Bounded	✓	✓	✓
[139]	Linux - TBS Qdisc	Predictable			
[138]	Linux - PREEMPT-RT	Bounded			
[53]	Virtual Switching	Bounded		✓	
[99]	Virtual Switching	Bounded	✓		
[159]	Virtual Switching	Bounded	✓		

[1] Solutions that provide predictable latency can guarantee a stable, exact latency value. This means also low jitter, hence, solutions with predictable latency can be used for deterministic communication.
[2] In order to exclude spatial interference between applications of different safety-criticalities, the typical solution is to provide applications with dedicated resources to rule out unintended resource interference.
[3] High availability and reliability ($\geq 99.999\%$).

A. Deterministic Networking

Over recent years, a large set of real-time fieldbus protocols were introduced in industrial networks [47], resulting in fragmented segments with proprietary communication solutions. One step towards the convergence in the industrial ecosystem was the usage of the real-time Ethernet variants, such as the old RTnet [73], or more recently EtherCAT [60] and PROFINET [40], that can provide deterministic performance, e.g. guaranteed, bounded latency. TSN, specified by IEEE 802.1 [141], seems now to be a converged, standardized Ethernet extension, introducing time-sensitive support. Hence, TSN is expected to replace the different, legacy real-time Ethernet and fieldbus variants. TSN includes a set of independent standards, which can be applied as a toolset to ensure a deterministic, reliable communication solution for time-critical applications. The IEEE 802.1AS defines the Generalized Precision Time Protocol (gPTP), which ensures that the clocks of each end-device can be synchronized with a μs-order precision.

The IEEE 802.1Qbv time-aware scheduling was designed to provide cyclical, pre-defined communication time slices for the different Ethernet QoS classes. A transmission gate

is associated with each queue belonging to a QoS class, and the queued frames can be selected for transmission only when the state of the gate is open. The configuration plan for the gates (gate control list) is calculated centrally, by the Central Network Configuration (CNC), based on the service requirements and the network characteristics (such as topology, TSN bridge delay, etc.), then the scheduling plan is pushed to each TSN bridge. The proper operation of 802.1Qbv requires time synchronization, which ensures that each device can handle the frames belonging to a certain QoS class in a network-wide, harmonized way. To reduce the end-to-end latency, it is important to move the control/server application instances closer to the client and the edge. The fog computing paradigm/architecture is a good candidate to support this goal.

In [113], TSN is proposed as the ideal choice for the networking layer of a fog computing-enabled industry automation deployment. The IEEE 802.1Qbv is recommended for handling the time-critical traffic and the related scheduling configuration challenges are overviewed in detail. It is assumed that Industry 4.0 applications require the frequent, dynamic reconfiguration of production processes, and consequently, the re-design of the traffic schedule is often needed. Therefore, a scheduling heuristic is proposed and evaluated, which can be used to reconfigure the transmission schedule at application runtime. A crucial goal in the heuristic is to avoid the interference between the concurrent time-critical streams using the minimum number of queues.

More recently, in [57], an architecture is proposed to integrate deterministic TSN networking within a Kubernetes-based container management infrastructure, called KuberneTSN. The concept is implemented in a seamless way within a TSN plug-in compliant with the Container Networking Interface (CNI), which uses also DPDK-based communications as available in OpenvSwitch (OVS) to achieve the lowest possible networking latency on the node. Experimental results with synthetic benchmarks show good potential for achieving lower and more stable networking latency, albeit the gain is in the range of tens of microseconds, in the considered set-up. The KuberneTSN framework is also used in [119] as a basis to build a "vPLC", i.e., a containerized equivalent of a programmable logic controller (PLC) handed over to a local private cloud, designed with in mind compatibility with open standards for Industry 4.0.

Many Industry 4.0 use cases require wireless connectivity to server mobile devices. In this case, 5G networks promise to be the best option for this purpose. However, the basic 5G URLLC feature [131] in itself is not enough to ensure the required determinism in communications for serving some time-critical Industrial 4.0 use cases, especially where time-awareness is crucial. Therefore, the 3rd Generation Partnership Project (3GPP) started to specify the Time Sensitive Communication (TSC) service for deterministic/synchronous communications in Release 16, and further enhancements are introduced in Release 17. The 3GPP Technical Specification (TS) 23.501 [137] describes the architecture for providing the seamless integration of 5G networks into the TSN ecosystem, describing how the 5G System is modeled as a virtual TSN bridge, the details of the control plane interworking, as well as how the characteristics of time-sensitive communications can be used to optimize the radio scheduling in the 5G domain.

A feasibility study on how a containerized, standard-compliant implementation of the 5G network can support the Generic Object Oriented Substation Event (GOOSE) protocol defined by the IEC 61850 standard for time-critical communications in Smart Grid is performed in [104]. Authors show that the average one-way delay between two endpoints is less than 0.6 ms, the average delay of the virtualized core network segment is 0.3 ms, while the maximum is below 0.4 ms (during the measurement period). The conclusion is that 5G network is able to support the time-critical GOOSE traffic. However, the radio transmission - which may have a significant effect on the end-to-end delay - is only emulated in the experiment, so this is a limitation of the work.

In [81], a detailed investigation is presented on how the current framework specified by the 3GPP for 5G and TSN integration can support deterministic communications; the paper also presents a quantitative analysis using a closed-loop control application. The paper shows a network scenario where the 5G domain is integrated into a wired TSN domain as a virtual TSN bridge (according to the 3GPP specification), and the packet delay budget and required bitrate for different deterministic, periodic traffic flows are calculated. The work identifies some design principles, such as: only on-premises 5G deployment can fulfill the strict latency requirements; and the adequate Radio Access Network (RAN) resource allocation is crucial, which can be achieved by using (semi-)persistent scheduling on the radio. It is noted that the estimation of the 5G virtual bridge delay is a difficult task, albeit this is one cornerstone of the 5G-TSN integration.

In [61], a scenario of cooperative work of mobile robots is presented, where tight synchronization is an essential requirement. Therefore, a concept is introduced on how the TSN time synchronization (IEEE 802.1AS) can be integrated with 5G networks in a seamless way. The essence is to use the synchronization between the gNodeB and the corresponding User Equipments (UEs), since this ensures the required time accuracy, and then calculate the offset between the TSN and 5G system clocks and adjust the local clocks in the UEs accordingly. It should be mentioned that, even if the proposed concept is validated on a 4G testbed, the measurement results confirm that the achieved accuracy is almost the same as the 802.1AS in a wired network deployment.

B. Real-time Communication Stack

The current trend in cloud application development is moving towards a cloud-native and microservices-based design: instead of deploying a monolithic application, its functionality is split into a multitude of micro-services deployed independently, so that they can exploit to the maximum extent the benefits of being deployed in a cloud environment. Therefore, it becomes paramount to use the appropriate networking abstraction among instances. For example, when using Kubernetes, several implementations of the CNI are available, with a wide variety of performance characteristics [115]. Moreover, in order to ensure the real-time communications for cloudified

applications, the real-time support features of the network stack in the virtualized domain (Linux communication stack, container networking) has to be leveraged. Considering the support of time-critical traffic (e.g. TSN 802.1Qbv scheduled traffic support) in an end-to-end manner, a software solution is required in the end hosts that are compatible with the 802.1Qbv traffic scheduling.

In [12], the implementation of a time-aware shaper using the Time-based Scheduling (TBS) Linux Qdisc [139] is presented, showing how to guarantee the schedule of packets to be transmitted. A detailed analysis is performed using a small demo network with 3 TSN bridges. The results show that, if TBS Qdisc is applied, the minimum transmission time is slightly increased, but the jitter could be two orders of magnitude lower than without TBS. The adjustment of the so-called minimum transmit offset, which is the time interval into the future the host application sets the packet transmit time in order to ensure that the packet is sent out in its scheduled time window, is also investigated. The measurements show that, if two hosts are directly interconnected, the transmission time increase is 55% due to the required transmit offset. However, in the case of a 3-bridge network, the increase is only 17.6%. To sum up, the support of time-critical applications requires using TBS Qdisc, together with a transmission offset, to handle the uncertainties of e.g., the CPU load due to the software stack. Although this causes some E2E delay increase, TSN-grade traffic scheduling is not possible otherwise.

Despite the fact that the Linux communication network subsystem is not optimized for bounded latency, a reasonably deterministic behavior is expected by using the PREEMPT-RT patch [138]. In [107], a comprehensive theoretical and experimental analysis of the Real-time Linux scheduling latency can be found, where the goal is to determine the sources of the latency. In [64], the real-time performance of Linux is reviewed with the usage of PREEMPT-RT in different system configurations. The results show that, although the non-real-time kernel exhibits the best average performance, a huge number of frames exceeds their latency bound, even if there is no background traffic. With PREEMPT-RT, a bounded latency can be achieved even in presence of concurrent traffic, but some missed deadlines can still be detected. Pinning the real-time tasks and the Interrupt Request (IRQ) of the real-time traffic queue to the same CPU, ensures a bounded latency in a stressed system, with no missed deadlines. If the real-time tasks are performed in an exclusively dedicated CPU, a similar behavior was detected, but with no further performance improvements. The main conclusion is that PREEMPT-RT is required for real-time applications, but in itself, it is not enough to guarantee bounded latency for each packet, so other system configurations, such as CPU isolation for real-time processes, are also required.

The authors in [77] emphasize that to execute collaborative tasks using a distributed control system, a large number of individual clocks in the system must be synchronized, and it should also be investigated how the time-based distributed scheduling of the control instances can be performed. The paper discusses in detail the support of the time synchronization in Linux, then measurements are performed using a set of TI Sitara am335x embedded System-on-Chip (SOC) devices. Authors describe several implementation options for real-time scheduling on Linux for SOC hardware. The main conclusions are that the most precise solution can be achieved if the system contains Programmable Real-time Unit (PRU), since in this case sub 1 μs scheduling accuracy and precision can be guaranteed. However, it should be noted that this solution requires special programming skills about the PRUs, and the code is not portable. Other options, such as HRTIMER service and SCHED-FIFO can provide only 100 μs scheduling accuracy which certainly limits the number of possible applications; the authors favor the simpler SCHED-FIFO based solution for applications with less stringent requirements.

The goal of [84] is to give a solution for the main performance challenge of virtualized environments, i.e., isolating a VM running a time-sensitive (e.g., soft real-time) application and a VMs running a bandwidth-intensive application, deployed on the same physical host. The paper shows the details of the transmission/reception as well as the essence of Qdisc capabilities in standard Linux and discusses the details of how the network stack is modified in Xen, considering two representative versions of Linux, e.g., Linux 3.10 and 3.18. From the perspective of prioritizing the time-sensitive traffic, two main limitations of the traffic control in Xen are identified, namely priority inversion between packets originating from the latency-sensitive and bandwidth-intensive domain, and priority inversion between the transmission and reception code paths. The main contribution of the paper is to introduce a Virtualization-aware Traffic Control (VATC) scheme, which ensures that time-sensitive flows coming from a VM are treated as high-priority traffic and assigned to high-priority kernel threads, so the real-time traffic can be protected; the number of priority levels can be configured according to the traffic characteristics. The paper also contains a detailed evaluation of VATC using various numbers of bandwidth-intensive background processes, showing that VATC can successfully mitigate the traffic control limitations of Xen.

The authors in [80] investigate a practical approach to ensure low latency communications within a distributed, virtualized 5G RAN deployment by introducing UDPDK, a novel open-source middleware on top of the Data Plane Development Kit (DPDK), for easing integration of DPDK-based communications. The authors also integrated UDPDK within the OpenAirInterface (OAI) software stack for RAN packet processing. The experimental results show that UDPDK can outperform a standard socket-based Application Programming Interface (API), achieving on average one-tenth of the Round-trip Time (RTT).

Another important aspect for supporting soft real-time services in a virtualized environment is the proper provisioning and management of the virtualized cloud resources considering the real-time constraints. On one hand, authors in [86] formulate the resource provisioning problem using a generic cloud model, a multi-tenant ecosystem, and a service chain deployment. On the other hand, an Integer Linear Programming (ILP)-based solution is proposed, called Network Functions Virtualization - Real-time (NFV-RT) whose main goal is to maximize the number of service requests by the tenants

meeting their bounded latency requirements, while VM-level isolation among tenants is ensured. The proposed algorithm performs the initial resource provisioning by determining the required number of service instances for a set of requests, as well as handling the new requests dynamically. The proposed method is evaluated by using simulations (comparing to a greedy heuristic) and using a real testbed. The results show a good performance ofNFV-RT. However, authors observe that the design of NFV-RT in reality guarantees that just about 94% of the packets belonging to an accepted service request meet the target deadline. Therefore, it can be used to provision virtualized network resources only for applications with soft real-time requirements.

C. Virtual Switching

Virtual switches are a cornerstone of today's virtualized data centers to enable communications among VMs, containers and also for communications across virtual and physical network segments. As mentioned above, time-sensitive communications can be supported by IEEE TSN-aware physical switches (considering especially IEEE 802.1Qbv and 802.1Qci). However, realizing a virtual switch that is TSN-enabled is still an open research challenge.

In [53], a deterministic message-switching solution is proposed by leveraging the Software Defined Networking (SDN) and TSN paradigms. After the discussion of the requirements of a time-aware virtual switch, the authors propose a detailed system model, where the synchronized operation, bounded relay overhead, and dedicated queues for each message at a specific point in time at the egress ports are emphasized, then the dispatching algorithm is described. A proof-of-concept implementation is also performed, where two virtual switches are deployed and a dedicated (isolated) CPU core is assigned to each of them; the virtual switch is implemented as a kernel module by leveraging the Linux Real-Time Application Interface (RTAI) patch. The overall latency caused by the virtual switches is observed in the range of 0.5 μs to 2 μs, but several outlier values can also be detected - the major reason behind this was the jitter caused by the RTAI scheduler.

Due to the strict latency requirements, the NFV industry is considering container-based deployments instead of traditional VMs. A detailed comparative study of the different software-based networking solutions for interconnecting VNF components in a private cloud infrastructure is presented in [18]. Starting with the overview of the different approaches of inter-container networking (e.g., Kernel-based networking, Kernel bypass, software switches) a detailed experimental analysis is performed using different test setups (intra-host and inter-host container networking). In the analysis, different virtual switch options (OVS, VPP, VALE, DPDK Basic Forwarding Sample Application, SR-IOV) are compared considering latency, throughput, and scalability performance factors. The conclusion is that SR-IOV and VALE outperform the other solutions on a single host not only in the performance parameters but also in terms of the processing power needed to perform high-speed packet forwarding. Considering communication between multiple hosts, the major bottleneck is caused by the CPU limitation and the throughput limitation of the underlying physical layer.

ESwitch [99] is a novel virtual switch architecture that breaks with general-purpose datapath and embraces a fully customized data plane. ESwitch exploits the fact that most real-world OpenFlow applications compose the same small set of simple, but generic forwarding patterns. So, OpenFlow pipelines can be rewritten in terms of a small set of static templates and then ESwitch uses a template-based code generation to derive the customized datapath. A detailed analysis shows that the ESwitch concept has significant performance gain over OVS, ensuring several times higher raw packet rates, much smaller latency, and predictable throughput even in case of varying workloads.

In [143] and [144], an extensible evolved Berkeley Packet Filter (eBPF)-based datapath architecture for OVS is proposed. The authors present the OVS and the eBPF forwarding models, then discuss the design and implementation of the OVS datapath considering the restrictions of eBPF followed by a brief performance analysis. More recently, an experimental comparison [111] in building network functions for packet processing in user-space versus kernel-space has been performed, using the recent `AF_XDP` socket type and eBPF, as well as recurring to poll-based processing.

One challenge in data-center networks is to provide a proper load-balancing scheme, which ensures that there is no congestion on some links, while others are underutilized. This problem is addressed in [159], which proposes Virtual Multi-channel Scatter (VMS), a packet load balancing solution that works in the virtual switch layer. The paper discusses in detail the possible load balancing schemes from the viewpoint of deployment location (e.g., centralized, end-host-based), and granularity (e.g., flow-level, packet-level) and concludes that the packet-level granularity should be leveraged to achieve load balancing in the virtual switch layer. The essence of VMS is to not distribute traffic evenly between available paths (channels) but applies a virtual window size estimation for each channel and to select channels accordingly. VMS can also handle the two main challenges of packet-level load-balancing; packet re-ordering and topology asymmetry-aware traffic distribution. Two implementation alternatives are presented, one based on OVS, and one based on SmartNIC. The evaluation shows that a near-optimal traffic distribution can be achieved even with an asymmetric topology, with tolerable latency and CPU utilization overheads in both implementations.

D. Discussion

In the communication area, the TSN technology and the 3GPP specified 5G-TSN integration ensure support for time-critical services in both the wired and wireless network domains. Numerous papers overview the real-time features of the Linux communication stack, which appears as a continuously evolving toolset capable of guaranteeing a certain level of timeliness in the cloud networking domain. However, by itself, it is not enough to provide a deterministic E2E communication: the integrated/harmonized operation of the different networking functions must be ensured through

multiple domains, but this area has not been thoroughly explored yet. For instance, harmonization and integration of traffic scheduling and control is needed at the hypervisor/host and guest/container levels. Moreover, handling of compute resources must also be harmonized with traffic scheduling, in order to guarantee that the application deployed in a container is able to generate a certain frame at the proper time, to ensure that it does not miss the target time window.

VI. Conclusions

The industrial and research communities are making steps forward in the various technological areas related to cloud and distributed computing. The acceleration of industrial digitalization and the corresponding emerging use cases are driving the demand for time-critical applications hosted in cloud environments. This paper surveyed the current state of the art concerning time-criticality in the cloud and the features to support time-critical applications. We provided a summary of key challenges in designing cloud infrastructures for time-critical services, covering aspects related to compute, orchestration, storage, and communications. Concerning the two questions posed at the beginning of the paper about the state of time-criticality in cloud computing, we can conclude that:

1) The main challenge for time-criticality in cloud technologies is ensuring temporal protection in presence of multi-tenancy. Most cloud services and infrastructures are still designed only to provide best-effort QoS. The focus is on maximizing overall throughput, neglecting the issue of providing predictable response times to the individual requests. However, incorporating time predictability properties into the cloud stack is essential to meet the requirements of many emerging time-critical applications (see Table I).

2) There are several solutions for high reliability and time-predictability within each domain. For example, recent developments in networking technologies, are becoming increasingly deterministic, with solutions like URLLC and TSN. However, there is a general lack of cross-domain efforts: for instance, despite the existence of hypervisors providing predictable CPU scheduling, no orchestration tools utilize such parameters to properly orchestrate the placement and resource provisioning of time-critical workloads, as briefly discussed in Section II-D. Another example is the one of real-time processing platforms: most of them focus on efficient process placement and resource allocation without taking into account storage-related matters, such as controlling the I/O backlog.

Regarding possible future directions, we highlighted that most of the academic papers found in the literature focus on low-latency, and only a few provide methods to guarantee a certain latency with high probability in distributed systems. Moreover, very few papers (mostly in the compute area) are backed up by real-time theory. In a real-world, time-critical scenario, a theoretical foundation is essential to assess the practicality of a time-critical solution. Hence the need for further investigations towards more formal evaluations of timing properties and guarantees. A promising research direction lies in the storage domain, which remains the most neglected area in modern literature, despite the existence of DynamoDB, the only cloud service with reasonably predictable performance guarantees.

In conclusion, no work in the literature proposes a comprehensive cloud system that incorporates time-critical guarantees within the entire cloud stack. Similarly, no commercial clouds integrate real-time configurations and, therefore, there are no ready-to-use time-critical cloud solutions yet.

References

[1] Luca Abeni, Alessio Balsini, and Tommaso Cucinotta. "Container-Based Real-Time Scheduling in the Linux Kernel". In: *ACM SIGBED Review* 16.3 (2019), pp. 33–38.

[2] Luca Abeni, Alessandro Biondi, and Enrico Bini. "Hierarchical Scheduling of Real-time Tasks over Linux-based Virtual Machines". In: *Journal of Systems and Software* 149 (2019), pp. 234–249.

[3] Luca Abeni and Giorgio Buttazzo. "Integrating Multimedia Applications in Hard Real-Time Systems". In: *IEEE Real-Time Systems Symposium*. RTSS '98. USA: IEEE Computer Society, 1998, p. 4.

[4] Luca Abeni and Dario Faggioli. "An Experimental Analysis of the Xen and KVM Latencies". In: *IEEE 22nd International Symposium on Real-Time Distributed Computing*. Valencia, Spain: IEEE, 2019, pp. 18–26.

[5] Luca Abeni and Dario Faggioli. "Using Xen and KVM as Real-Time Hypervisors". In: *Journal of Systems Architecture* 106 (2020), p. 101709.

[6] Luca Abeni et al. "A Measurement-Based Analysis of the Real-Time Performance of Linux". In: *8th IEEE Real-Time and Embedded Technology and Applications Symposium*. USA: IEEE, 2002, pp. 133–142.

[7] Luca Abeni et al. "Fault Tolerance in Real-Time Cloud Computing". In: *IEEE 26th International Symposium On Real-Time Distributed Computing*. IEEE. 2023.

[8] 5G ACIA White Paper. *5G for Connected Industries and Automation - Second Edition*. 2019.

[9] Mahbuba Afrin et al. "Resource Allocation and Service Provisioning in Multi-Agent Cloud Robotics: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 23.2 (2021), pp. 842–870.

[10] Alexandru Agache et al. "Firecracker: Lightweight Virtualization for Serverless Applications". In: *17th USENIX Symposium on Networked Systems Design and Implementation*. 2020, pp. 419–434.

[11] Marcos K. Aguilera et al. "Microsecond Consensus for Microsecond Applications". In: *14th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, Nov. 2020, pp. 599–616.

[12] Syed Sahal Nazli Alhady and Amir Fuad Wajdi Othman. "Time-Aware Traffic Shaper Using Time-based Packet Scheduling on Intel I210". In: *International Journal of Research and Engineering* 5.9 (2018), pp. 494–499.

[13] Fredrik Alriksson et al. "Critical IoT connectivity Ideal for Time-Critical Communications". In: *Ericsson Technology Review* 2020.6 (2020), pp. 2–13.

[14] Fredrik Alriksson et al. "XR and 5G: Extended reality at scale with time-critical communication". In: *Ericsson Technology Review* 2021.8 (Aug. 2021), pp. 2–13.

[15] Remo Andreoli, Tommaso Cucinotta, and Daniel Bristot De Oliveira. "Priority-Driven Differentiated Performance for NoSQL Database-As-a-Service". In: *IEEE Transactions on Cloud Computing* 11.4 (2023), pp. 3469–3482.

[16] Remo Andreoli et al. "Design-Time Analysis of Time-Critical and Fault-Tolerance Constraints in Cloud Services". In: *IEEE 16th Int. Conf. on Cloud Computing*. Chicago, USA: IEEE, 2023, pp. 415–417.

[17] Remo Andreoli et al. “Optimal Deployment of Cloud-native Applications with Fault-Tolerance and Time-Critical End-to-End Constraints”. In: *Proc. 16th IEEE/ACM Intl. Conf. on Cloud Computing*. IEEE. Taormina, Messina, Italy, 2023.

[18] Gabriele Ara et al. “A Framework for Comparative Evaluation of High-Performance Virtualized Networking Mechanisms”. In: *Cloud Comp. and Services Science*. Ed. by Donald Ferguson, Claus Pahl, and Markus Helfert. Vol. 1399. Cham: Springer International Publishing, 2021, pp. 59–83.

[19] Michael Armbrust et al. “A View of Cloud Computing”. In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[20] Michael Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. Univ. of California, 2009.

[21] Shehzad Ali Ashraf et al. *5G-powered FRMCS*. Tech. rep. Ericsson White Paper, Mar. 2022.

[22] Ali Balador et al. “AORTA: Advanced Offloading for Real-time Applications”. English. In: July 2023.

[23] Ioana Baldini et al. “Serverless Computing: Current Trends and Open Problems”. In: *Research Advances in Cloud Computing*. Ed. by Sanjay Chaudhary, Gaurav Somani, and Rajkumar Buyya. Springer Singapore, 2017, pp. 1–20.

[24] Marco Barletta et al. “Achieving Isolation in Mixed-Criticality Industrial Edge Systems with Real-Time Containers”. In: *34th Euromicro Conference on Real-Time Systems*. Vol. 231. Dagstuhl, Germany, 2022, 15:1–15:23.

[25] Marco Barletta et al. “Criticality-aware monitoring and orchestration for containerized industry 4.0 environments”. In: *ACM Trans. on Embedded Comp. Sys.* 23.1 (2024), pp. 1–28.

[26] Marco Barletta et al. “Failover Timing Analysis in Orchestrating Container-based Critical Applications”. In: *19th European Dependable Computing Conference*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2024, pp. 81–84.

[27] Marco Barletta et al. “Mutiny! How Does Kubernetes Fail, and What Can We Do About It?” In: *2024 54th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2024, pp. 1–14.

[28] Rasoul Behravesh, Estefania Coronado, and Roberto Riggio. “Performance Evaluation on Virtualization Technologies for NFV Deployment in 5G Networks”. In: *IEEE Conference on Network Softwarization*. Paris, France, 2019, pp. 24–29.

[29] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. “LightNVM: The Linux Open-Channel SSD Subsystem”. In: *15th USENIX Conference on File and Storage Technologies (FAST 17)*. 2017, pp. 359–374.

[30] Matias Bjørling et al. “ZNS: Avoiding the Block Interface Tax for Flash-based SSDs”. In: *USENIX Annual Technical Conference*. 2021, pp. 689–703.

[31] Marta Catillo, Umberto Villano, and Massimiliano Rak. “A survey on auto-scaling: how to exploit cloud elasticity”. In: *Int. J. Grid Util. Comput.* 14.1 (Jan. 2023), pp. 37–50.

[32] Rick Cattell. “Scalable SQL and NoSQL Data Stores”. In: *ACM Sigmod Record* 39.4 (2011), pp. 12–27.

[33] Huangke Chen et al. “Towards Energy-Efficient Scheduling for Real-Time Tasks under Uncertain Cloud Computing Environment”. In: *J. of Sys. and Softw.* 99 (2015), pp. 20–35.

[34] Kuan-Hsun Chen et al. “Unikernel-Based Real-Time Virtualization Under Deferrable Servers: Analysis and Realization”. In: *Proc. 34th Euromicro Conference on Real-Time Systems*. Vol. 231. Dagstuhl, Germany, 2022, 6:1–6:22.

[35] Shichao Chen and Mengchu Zhou. “Evolving Container to Unikernel for Edge Computing and Applications in Process Industry”. In: *Processes* 9.2 (Feb. 14, 2021), p. 351.

[36] Claudio Cicconetti et al. “Toward Distributed Computing Environments with Serverless Solutions in Edge Systems”. In: *IEEE Communications Magazine* 58.3 (2020), pp. 40–46.

[37] Marcello Cinque and Domenico Cotroneo. “Towards Lightweight Temporal and Fault Isolation in Mixed-Criticality Systems with Real-Time Containers”. In: *48th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks Workshops*. Luxembourg: IEEE, 2018, pp. 59–60.

[38] Marcello Cinque, Raffaele Della Corte, and Roberto Ruggiero. “Preventing Timing Failures in Mixed-criticality Clouds with Dynamic Real-time Containers”. In: *17th European Dependable Computing Conference*. Munich, Germany: IEEE, 2021, pp. 17–24.

[39] Marcello Cinque et al. “RT-CASEs: Container-Based Virtualization for Temporally Separated Mixed-Criticality Task Sets”. In: *Proc. 31st Euromicro Conference on Real-Time Systems*. Dagstuhl, Germany, 2019, 5:1–5:22.

[40] PROFINET consortium. *PROFINET – The Solution Platform for Process Automation*. 2018.

[41] A. Crespo, I. Ripoll, and M. Masmano. “Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach”. In: *2010 European Dependable Computing Conference*. 2010, pp. 67–72.

[42] Tommaso Cucinotta et al. “A Real-Time Service-Oriented Architecture for Industrial Automation”. In: *IEEE Transactions on Industrial Informatics* 5.3 (Aug. 2009), pp. 267–277.

[43] Tommaso Cucinotta et al. “Strong Temporal Isolation among Containers in OpenStack for NFV Services”. In: *IEEE Transactions on Cloud Computing* 11.1 (2023), pp. 763–778.

[44] Tommaso Cucinotta et al. “The Importance of Being OS-aware – In Performance Aspects of Cloud Computing Research”. In: *8th Int. Conf. on Cloud Computing and Services Science*. SciTePress, 2018, pp. 626–633.

[45] Tommaso Cucinotta et al. “The IRMOS/ISONI Real-Time Cloud Infrastructure: a Virtualised e-Learning Case-Study”. In: *IEEE Multimedia Comm. Techn. Committee* 6.12 (2011).

[46] Tommaso Cucinotta et al. “Virtualised e-Learning with Real-time Guarantees on the IRMOS platform”. In: *IEEE Int. Conf. on Service-Oriented Comp. and Appl.* 2010, pp. 1–8.

[47] Peter Danielis et al. “Real- Time Capable Internet Technologies for Wired Communication in the Industrial IoT-a Survey”. In: *IEEE 23rd Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. Oct. 2018, pp. 266–273.

[48] Jakob Danielsson, Nandinbaatar Tsog, and Ashalatha Kunnappilly. “A Systematic Mapping Study on Real-Time Cloud Services”. In: *IEEE/ACM Int. Conf. on Utility and Cloud Computing Companion*. 2018, pp. 245–251.

[49] Giuseppe DeCandia et al. “Dynamo: Amazon’s Highly Available Key-Value Store”. In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007), pp. 205–220.

[50] ETSI. *Network Functions Virtualisation*. White Paper 1. Darmstadt: SDN and Openflow World Congress, 2012.

[51] Hao Fan et al. “Accelerating Parallel Applications in Cloud Platforms via Adaptive Time-Slice Control”. In: *IEEE Transactions on Computers* 70.7 (2020), pp. 992–1005.

[52] Hao Fan et al. “Gear: Enable Efficient Container Storage and Deployment with a New Image Format”. In: *IEEE 41st Int. Conf. on Distributed Computing Systems*. IEEE, 2021, pp. 115–125.

[53] Hongjie Fang and Roman Obermaisser. “Virtual Switch for Integrated Real-Time Systems based on SDN”. In: *6th Intl. Conf. on Internet of Things: Systems, Management and Security*. Granada, Spain: IEEE, 2019, pp. 344–351.

[54] Stefano Fiori, Luca Abeni, and Tommaso Cucinotta. “RT-Kubernetes: Containerized Real-time Cloud Computing”. In: *37th ACM/SIGAPP Symposium on Applied Computing*. Virtual Event: ACM, 2022, pp. 36–39.

[55] *First Report on 5G Network Architecture Options and Assessments*. Tech. rep. 5G SMART, 2020.

[56] Xinwei Fu et al. “EdgeWise: A Better Stream Processing Engine for the Edge”. In: *USENIX Annual Technical Conference*. Renton, WA, 2019, pp. 929–946.

[57] Andrea Garbugli et al. “KuberneTSN: a Deterministic Overlay Network for Time-Sensitive Containerized Environ-

ments". In: *ICC 2023 - IEEE Int. Conf. on Communications*. May 2023, pp. 1494–1499.

[58] Marisol García-Valls, Tommaso Cucinotta, and Chenyang Lu. "Challenges in real-time virtualization and predictable cloud computing". In: *Journal of Systems Architecture* 60.9 (2014), pp. 726–740.

[59] Jinkun Geng et al. "Nezha: Deployable and High-Performance Consensus Using Synchronized Clocks". In: *Proc. VLDB Endow.* 16.4 (Dec. 2022), pp. 629–642.

[60] EtherCAT Technology Group. *EtherCAT – The Ethernet Fieldbus*. Mar. 2020.

[61] Michael Gundall et al. "Integration of 5G with TSN as Prerequisite for a Highly Flexible Future Industrial Automation: Time Synchronization based on IEEE 802.1AS". In: *The 46th Annual Conference of the IEEE Industrial Electronics Society*. Singapore, Singapore: IEEE, 2020, pp. 3823–3830.

[62] Michael Gundall et al. "Introduction of a 5G-Enabled Architecture for the Realization of Industry 4.0 Use Cases". In: *IEEE Access* 9 (2021), pp. 25508–25521.

[63] Harshit Gupta and Umakishore Ramachandran. "Fogstore: A Geo-Distributed Key-Value Store Guaranteeing Low Latency for Strongly Consistent Access". In: *12th ACM Int. Conf. on Distributed and Event-based Systems*. 2018, pp. 148–159.

[64] Carlos San Vicente Gutiérrez et al. "Real-time Linux Communications: An Evaluation of the Linux Communication Stack for Real-time Robotic Applications". In: *ArXiv* abs/1808.10821 (2018).

[65] H. Li, C. Lu, and C. Gill. "Predicting Latency Distributions of Aperiodic Time-Critical Services". In: *IEEE Real-Time Systems Symposium*. 2019, pp. 30–42.

[66] Moin Hasan and Major Singh Goraya. "Fault Tolerance in Cloud Computing Environment: A Systematic Survey". In: *Computers in Industry* 99 (2018), pp. 156–172.

[67] Florian Hofer et al. "Industrial Control via Application Containers: Migrating from Bare-Metal to IAAS". In: *IEEE Int. Conf. on Cloud Computing Technology and Science*. 2019, pp. 62–69.

[68] Eric Jonas et al. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Tech. rep. Univ. of California, Berkeley, Feb. 2019.

[69] Kyoung-Don Kang, Jisu Oh, and Sang H. Son. "Chronos: Feedback Control of a Real Database Sys. Perf." In: *28th IEEE Int. Real-Time Systems Symp.* 2007, pp. 267–276.

[70] Woochul Kang, Sang H. Son, and John A. Stankovic. "QeDB: A Quality-Aware Embedded Real-Time Database". In: *15th IEEE Real-Time and Embedded Technology and Applications Symposium*. 2009, pp. 108–117.

[71] Ben Kao and Hector Garcia-Molina. "An Overview of Real-Time Database Systems". In: *Real Time Computing* (1994), pp. 261–282.

[72] Chesta Kathpal and Ritu Garg. "Survey on Fault-Tolerance-Aware Scheduling in Cloud Computing". In: *Information and Communication Technology for Competitive Strategies*. Springer, 2019, pp. 275–283.

[73] J. Kiszka and B. Wagner. "RTnet - a flexible hard real-time networking framework". In: *2005 IEEE Conference on Emerging Technologies and Factory Automation*. Vol. 1. Oct. 2005, 8 pp.–456.

[74] Gerwin Klein et al. "seL4: formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 207–220.

[75] Marios Kogias and Edouard Bugnion. "HovercRaft: Achieving Scalability and Fault-Tolerance for Microsecond-Scale Datacenter Services". In: *Fifteenth European Conference on Computer Systems*. 2020, pp. 1–17.

[76] Kleopatra Konstanteli et al. "Elastic Admission Control for Federated Cloud Services". In: *IEEE Transactions on Cloud Computing* 2.3 (2014), pp. 348–361.

[77] Tamas Kovacshazy, Tibor Tusori, and David Vincze. "Prototype Implementation and Performance of Time-based Distributed Scheduling on Linux for Real-Time Cyber-Physical Systems". In: *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication*. Geneva: IEEE, Sept. 2018, pp. 1–6.

[78] Tytus Kurek. "Unikernel Network Functions: A Journey Beyond the Containers". In: *IEEE Commun. Mag.* 57.12 (Dec. 2019), pp. 15–19.

[79] Dimosthenis Kyriazis, Theodora A. Varvarigou, and Kleopatra Konstanteli. "Achieving Real-Time in Distributed Computing - From Grids to Clouds". In: *Achieving Real-Time in Distributed Computing*. 2011.

[80] Leonardo Lai et al. "Ultra-low Latency NFV Services Using DPDK". In: *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Heraklion, Greece: IEEE, Nov. 2021, pp. 8–14.

[81] Ana Larranaga et al. "Analysis of 5G-TSN Integration to Support Industry 4.0". In: *25th IEEE Int. Conf. on Emerging Technologies and Factory Automation*. Vienna, Austria: IEEE, Sept. 2020, pp. 1111–1114.

[82] Gyusun Lee, Seokha Shin, and Jinkyu Jeong. "Efficient Hybrid Polling for Ultra-low Latency Storage Devices". In: *Journal of Systems Architecture* 122 (2022), p. 102338.

[83] Min Lee et al. "Supporting Soft Real-time Tasks in the Xen Hypervisor". In: *6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. Pittsburgh, Pennsylvania, USA: ACM Press, 2010, p. 97.

[84] Chong Li et al. "Prioritizing Soft Real-time Network Traffic in Virtualized Hosts based on Xen". In: *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. Seattle, WA, USA: IEEE, Apr. 2015, pp. 145–156.

[85] Hao Li et al. "ACRN: A Big Little Hypervisor for IoT Development". In: *15th Int. Conf. on Virtual Execution Environments*. Providence, RI, USA: ACM, 2019, pp. 31–44.

[86] Yang Li, Linh Thi Xuan Phan, and Boon Thau Loo. "Network Functions Virtualization with Soft Real-time Guarantees". In: *35th Annual Int. Conf. on Computer Communications*. San Francisco, CA: IEEE, Apr. 2016, pp. 1–9.

[87] Youhuizi Li et al. "PINE: Optimizing Perf. Isolation in Container Env." In: *IEEE Access* 7 (2019), pp. 30410–30422.

[88] J. Liedtke, H. Hartig, and M. Hohmuth. "OS-controlled cache predictability for real-time systems". In: *Proceedings Third IEEE Real-Time Technology and Applications Symposium*. 1997, pp. 213–224.

[89] Jan Lindström. "Optimistic Concurrency Control Methods for Real-Time Database Systems". PhD thesis. 2003.

[90] Heiner Litz et al. "RAIL: Predictable, Low Tail Latency for NVMe Flash". In: *ACM Transactions on Storage* 18.1 (2022), pp. 1–21.

[91] Francesco Lumpp et al. "Enabling Kubernetes Orchestration of Mixed-Criticality Softw. for Autonomous Mobile Robots". In: *IEEE Trans. on Robotics* 40 (2024), pp. 540–553.

[92] Arlino Magalhaes, Jose Maria Monteiro, and Angelo Brayner. "Main Memory Database Recovery: A Survey". In: *ACM Comput. Surv.* 54.2 (Mar. 2021).

[93] Chen-Nien Mao et al. "Minimizing Latency of Real-Time Container Cloud for Software Radio Access Networks". In: *7th Intern. Conf. on Cloud Computing Technology and Science*. Vancouver, BC, Canada: IEEE, 2015, pp. 611–616.

[94] José Martins et al. "Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems". In: *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*. Vol. 77. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 3:1–3:14.

[95] Ilias Mavridis and Helen Karatza. "Lightweight Virtualization Approaches for Software-Defined Systems and Cloud Computing: An Evaluation of Unikernels and Containers". In: *Sixth Int. Conf. on Software Defined Systems*. Rome, Italy: IEEE, 2019, pp. 171–178.

[96] Ilias Mavridis and Helen Karatza. "Orchestrated Sandboxed Containers, Unikernels, and Virtual Machines for Isolation-enhanced Multitenant Workloads and Serverless Computing in Cloud". In: *Concurrency Computat Pract Exper* (2021).

[97] Sambit Kumar Mishra, Bibhudatta Sahoo, and Priti Paramita Parida. "Load Balancing in Cloud Computing: A Big Picture". In: *Journal of King Saud University - Computer and Information Sciences* 32.2 (Feb. 2020), pp. 149–158.

[98] Chetankumar Mistry et al. "Demonstrating the Practicality of Unikernels to Build a Serverless Platform at the Edge". In: *IEEE Int. Conf. on Cloud Computing Technology and Science*. Bangkok: IEEE, 2020, pp. 25–32.

[99] László Molnár et al. "Dataplane Specialization for High-performance OpenFlow Software Switching". en. In: *2016 ACM SIGCOMM Conference*. Florianopolis Brazil: ACM, Aug. 2016, pp. 539–552.

[100] Derek G. Murray et al. "Naiad: A Timely Dataflow System". In: *Proc. 24th ACM Symposium on Operating Systems Principles*. Farminton, Pennsylvania: ACM, 2013, pp. 439–455.

[101] Gábor Németh, Dániel Géhberger, and Péter Mátray. "DAL: A Locality-Optimizing Distrib. Shared Memory Sys." In: *9th USENIX Workshop on Hot Topics in Cloud Comp.* 2017.

[102] Hai Duc Nguyen and Andrew A. Chien. "Storm-RTS: Stream Processing with Stable Perf. for Multi-cloud and Cloud-Edge". In: *IEEE 16th Int. Conf. on Cloud Comp.* 2023.

[103] Hai Duc Nguyen et al. "Real-Time Serverless: Enabling Application Performance Guarantees". In: *5th International Workshop on Serverless Computing*. WOSC '19. Davis, CA, USA: ACM, 2019, pp. 1–6.

[104] Van-Giang Nguyen et al. "A Deployable Containerized 5G Core Solution for Time Critical Communication in Smart Grid". In: *23rd Conf. on Intelligence in Next Generation Networks*. Paris, France: IEEE, 2020, pp. 153–155.

[105] Joseph Noor, Mani Srivastava, and Ravi Netravali. "Portkey: Adaptive Key-Value Placement over Dynamic Edge Networks". In: *ACM Symposium on Cloud Computing*. Seattle, WA, USA: ACM, 2021, pp. 197–213.

[106] Andres F. Ocampo et al. "Opportunistic CPU Sharing in Mobile Edge Computing Deploying the Cloud-RAN". In: *IEEE Transactions on Network and Service Management* 20.3 (Oct. 2023), pp. 2201–2217.

[107] Daniel Bristot de Oliveira et al. "Demystifying the Real-Time Linux Scheduling Latency". In: *32nd Euromicro Conference on RT Systems*. Dagstuhl, Germany, 2020, 9:1–9:23.

[108] Luca Orciari, Davide Raggini, and Andrea Tilli. "Taming Edge Computing for Hard Real-Time Advanced Control of Mechatronic Systems". In: *IEEE Transactions on Industrial Informatics* 20.8 (Aug. 2024), pp. 9898–9906.

[109] Mark Panahi, Weiran Nie, and Kwei-Jay Lin. "A Framework for Real-Time Service-Oriented Architecture". In: *2009 IEEE Conference on Commerce and Enterprise Computing*. July 2009, pp. 460–467.

[110] 5G ACIA White Paper. *Integration of 5G with Time-Sensitive Networking for Industrial Communications*. 2021.

[111] Federico Parola et al. "Comparing User Space and In-Kernel Packet Processing for Edge Data Centers". In: *SIGCOMM Comput. Commun. Rev.* 53.1 (Apr. 2023), pp. 14–29.

[112] Somasundaram Perianayagam et al. "Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service". In: *USENIX Annual Technical Conference*. Carlsbad, CA, 2022, pp. 1037–1048.

[113] Paul Pop et al. "Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 55–61.

[114] Mia Primorac. "Understanding and Mitigating Latency Variability of Latency-Critical Applications". PhD thesis. Lausanne: IINFCOM, 2020, pp. 1–125.

[115] Shixiong Qi, Sameer G. Kulkarni, and K. K. Ramakrishnan. "Assessing Container Network Interface Plugins: Functionality, Performance, and Scalability". In: *IEEE Transactions on Network and Service Management* 18.1 (2021), pp. 656–671.

[116] Rui Queiroz et al. "Container-based Virtualization for Real-time Industrial Systems – A Systematic Review". In: *ACM Comput. Surv.* 56.3 (Oct. 2023).

[117] Ralf Ramsauer et al. "Look Mum, no VM Exits!(Almost)". In: *13th Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'17)*. Duprovnik, Croatia, June 2017.

[118] Arun Ravindran and Anjus George. "An Edge Datastore Architecture For Latency-Critical Distributed Machine Vision Applications". In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*. 2018.

[119] Lorenzo Rosa et al. "Supporting vPLC Networking over TSN with Kubernetes in Industry 4.0". In: *1st Workshop on Enhanced Network Techniques and Technologies for the Industrial IoT to Cloud Continuum*. New York, NY, USA: ACM, 2023, pp. 15–21.

[120] Mahadev Satyanarayanan. "The Emergence of Edge Computing". In: *Computer* 50.1 (2017), pp. 30–39.

[121] Johann Schleier-Smith et al. "What Serverless Computing is and Should Become: The next Phase of Cloud Computing". In: *Commun. ACM* 64.5 (Apr. 2021), pp. 76–84.

[122] Weisong Shi et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things J* 3 (2016), pp. 637–646.

[123] Insik Shin and Insup Lee. "Compositional Real-time Scheduling Framework". In: *25th IEEE International Real-Time Systems Symposium*. Dec. 2004, pp. 57–67.

[124] Insik Shin and Insup Lee. "Compositional Real-Time Scheduling Framework with Periodic Model". In: *ACM Trans. Embed. Comput. Syst.* 7.3 (May 2008).

[125] Garfinkel Simson L. and Abelson Harold. *Architects of the Information Society: Thirty-Five Years of the Laboratory for Computer Science at MIT*. The MIT Press, 1999.

[126] Per Skarin et al. "Control-over-the-cloud: A Performance Study for Cloud-native, Critical Control Systems". In: *2020 IEEE/ACM 13th Int. Conf. on Utility and Cloud Computing (UCC)*. 2020, pp. 57–66.

[127] V Srinivasan and Brian Bulkowski. "Citrusleaf: A Real-time NoSQL DB which Preserves ACID". In: *VLDB Endowment* 4.12 (2011), pp. 1340–1350.

[128] Václav Struhár et al. "Hierarchical Resource Orchestration Framework for Real-time Containers". In: *ACM Trans. Embed. Comput. Syst.* 23.1 (Jan. 2024).

[129] Václav Struhár et al. "REACT: Enabling Real-Time Container Orchestration". In: *26th IEEE Intl. Conf. on Emerging Technologies and Factory Automation*. 2021, pp. 1–8.

[130] Václav Struhár et al. "Real-Time Containers: A Survey". In: *Proc. 2nd Workshop on Fog Computing and the IoT*. Vol. 80. Dagstuhl, Germany, 2020, 7:1–7:9.

[131] *Study on enhancement of Ultra-Reliable Low-Latency Communication (URLLC) support in the 5G Core network (5GC)*. https://www.3gpp.org/DynaReport/23725.htm.

[132] Yu Sun et al. "Baoverlay: A Block-Accessible Overlay File System for Fast and Efficient Container Storage". In: *Proc. 11th ACM Symp. on Cloud Computing*. 2020, pp. 90–104.

[133] Róbert Szabó et al. "Elastic network functions: opportunities and challenges". In: *IEEE network* 29.3 (2015), pp. 15–21.

[134] Mark Szalay, Peter Matray, and Laszlo Toka. "Real-time faas: Towards a latency bounded serverless cloud". In: *IEEE Transactions on Cloud Computing* (2022).

[135] Márk Szalay, Péter Mátray, and László Toka. "Real-Time Task Scheduling in a FaaS Cloud". In: *IEEE 14th Intl. Conf. on Cloud Computing*. IEEE, 2021, pp. 497–507.

[136] Timur Tasci, Jan Melcher, and Alexander Verl. "A Container-Based Architecture for Real-Time Control Applications". In: *IEEE Int. Conf. on Engineering, Technology and Innovation*. IEEE, 2018, pp. 1–9.

[137] *The 3GPP Technical Specification (TS) 23.501 - Time Sensitive Communication (TSC)*. https : / / www . 3gpp . org / DynaReport/23501.htm.

[138] *The Linux Real Time Collaborative Project - PREEMPT_RT Patches*. https://wiki.linuxfoundation.org/realtime/start.

[139] *Time-based Scheduling (TBS) Linux Qdisc*. https : / / tsn . readthedocs.io/qdiscs.html.

[140] *Time-Critical Communication: Leading the Next Wave of 5G Innovation*. Tech. rep. Ericsson, 2021.

[141] *Time-Sensitive Networking (TSN) Task Group*. https : / / 1 . ieee802.org/tsn/.

[142] László Toka. "Ultra-Reliable and Low-Latency Computing in the Edge with Kubernetes". In: *Journal of Grid Computing* 19.3 (2021), pp. 1–23.

[143] Cheng-Chun Tu, Joe Stringer, and Justin Pettit. "Building an Extensible Open vSwitch Datapath". en. In: *ACM SIGOPS Operating Systems Review* 51.1 (Sept. 2017), pp. 72–77.

[144] William Tu et al. *Bringing the Power of eBPF to Open vSwitch*. Linux Plumbers Conference. 2018.

[145] Achilleas Tzenetopoulos et al. "Interference-aware orchestration in kubernetes". In: *International conference on high performance computing*. Springer. 2020, pp. 321–330.

[146] Mikko Uusitalo et al. *European Vision for the 6G Network Ecosystem*. Nov. 2024.

[147] Paolo Valente and Fabio Checconi. "High Throughput Disk Scheduling with Fair Bandwidth Distribution". In: *IEEE Transactions on Computers* 59.9 (2010), pp. 1172–1186.

[148] Richard West et al. "A Virtualized Separation Kernel for Mixed-Criticality Systems". In: *ACM Trans. Comput. Syst.* 34.3 (June 2016).

[149] Chenggang Wu, Vikram Sreekanti, and Joseph M Hellerstein. "Autoscaling Tiered Cloud Storage in Anna". In: *The VLDB Journal* 30.1 (2021), pp. 25–43.

[150] Jun Wu and Tung-I Yang. "Dynamic CPU Allocation for Docker Containerized Mixed-criticality Real-time Systems". In: *IEEE Int. Conf. on Applied System Invention*. Chiba: IEEE, 2018, pp. 279–282.

[151] Song Wu et al. "Container-Aware I/O Stack: Bridging the Gap between Container Storage Drivers and Solid State Devices". In: *18th ACM SIGPLAN/SIGOPS Intl. Conf. Virtual Execution Environments*. 2022, pp. 18–30.

[152] Sisu Xi et al. "RT-Open Stack: CPU Resource Management for Real-Time Cloud Computing". In: *2015 IEEE 8th Int. Conf. on Cloud Computing*. New York City, NY, USA: IEEE, June 2015, pp. 179–186.

[153] Sisu Xi et al. "RT-Xen: Towards Real-time Hypervisor Scheduling in Xen". In: *Ninth ACM Int. Conf. on Embedded Software*. 2011, pp. 39–48.

[154] Chen Xu et al. "AQUAS: A Quality-Aware Scheduler for NoSQL Data Stores". In: *IEEE 30th Int. Conf. on Data Engineering*. IEEE, 2014, pp. 1210–1213.

[155] Chen Xu et al. "Quality-Aware Schedulers for Weak Consistency Key-Value Data Stores". In: *Distributed and Parallel Databases* 32.4 (2014), pp. 535–581.

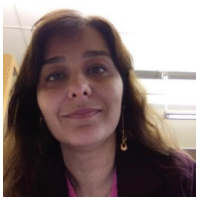
[156] Jinlai Xu et al. "Amnis: Optimized Stream Processing for Edge Computing". In: *Journal of Parallel and Distributed Computing* 160 (2022), pp. 49–64.

[157] Xun Xu. "From Cloud Comp. to Cloud Manuf." In: *Robotics and computer-integrated manuf.* 28.1 (2012), pp. 75–86.

[158] P.S. Yu et al. "On Real-time Databases: Concurrency Control and Scheduling". In: *Proc. IEEE* 82.1 (1994), pp. 140–157.

[159] Yiran Zhang et al. "VMS: Load Balancing Based on the Virtual Switch Layer in Datacenter Networks". In: *IEEE Journal on Selected Areas in Communications* 38.6 (June 2020), pp. 1176–1190.

Remo Andreoli has a MSc with honors in Computer Science from University of Pisa (Italy). He is currently a PhD student at Scuola Superiore Sant'Anna in Pisa (Italy). His research focuses on differentiated performance mechanisms for NoSQL databases and resource management optimization techniques for cloud infrastructures. He earned a best student paper award at CLOSER 2021.



Raquel Mini is a researcher at Ericsson working in the Cloud Systems and Platforms area. She got her BSc, MSc, and PhD in Computer Science from Federal University of Minas Gerais (UFMG), Brazil. Before joining Ericsson, Raquel worked for more than 20 years as a Professor of Computer Science in Brazil. Her research addresses the area of sensor networks, IoT, ubiquitous and cloud computing.

Per Skarin just recently joined SAAB as a Senior Research Scientist. He has been a researcher and Machine Learning Specialist at Ericsson for almost 11 years, working in the Cloud Technology and Autonomous Systems area. He has a PhD from Lund University (Sweden).

Harald Gustafsson is an Expert in Network-Compute Application Platforms at Ericsson Research, Lund (Sweden). He has a MSc in Electrical Engineering from Lund University, and a PhD in Applied Signal Processing from the Blekinge Institute of Technology (Sweden). His research addresses multimedia systems, distributed systems, communication systems and cloud systems on aspects as time criticality, reliability, and energy efficiency.

Janos Harmatos Janos Harmatos has a PhD in electrical engineering from the Budapest University of Technology and Economics, Hungary. He is currently a Master Researcher with Ericsson Research, investigating on deterministic communication and real-time cloud systems. He was involved in several standardization bodies, such as 3GPP, ETSI, and 5G-ACIA.

Luca Abeni is Associate Professor at Scuola Superiore Sant'Anna. He got a MSc in computer engineering from University of Pisa, and a PhD from Scuola Superiore Sant'Anna. He has also been researcher and Associate Professor at University of Trento. His main research interests are operating systems, scheduling algorithms, QoS management, multimedia applications, and audio/video streaming.

Tommaso Cucinotta is Associate Professor at Scuola Superiore Sant'Anna, where he got a PhD in Computer Engineering. He got also a MSc in Computer Engineering from University of Pisa. His research interests include predictability in Cloud Computing and NFV. He has been MTS in Bell Labs in Dublin (Ireland), investigating on security and real-time performance of cloud services.