Research Statement

Marios Kogias

Online services play a major role in our everyday life for communication, entertainment, socializing, e-commerce, *etc.*. The COVID-19 global pandemic and social distancing made those services even more fundamental. The heart of those online services is the datacenter and the performance of those services has vast economical, *e.g.,* tech giants' revenue stream, and sociological, *e.g.,* online user engagement and e-learning, effects. My research lies on the intersection of operating systems and networking and focuses on building efficient latency-critical datacenter systems. By revisiting existing mechanisms and designing new abstractions specifically for the datacenter environment I work towards a datacenter infrastructure that can provide strong tail-tolerance and fault-tolerance guarantess in the *microsecond (*µs*)* scale.

The datacenter environment differs drastically from other system domains, *e.g.,* mobile or edge computing. This environment creates several new challenges and opportunities for system design. The currently deployed datacenter hardware is capable of transmitting a minimal-sized packet in less than 10µs and accessing data from a hard disk in about 100µs. However, existing operating systems and networking stacks have not been designed for a µs-scale leading to inefficient use of hardware resources. Also, emerging programmable devices, such as programmable switches and NICs, start being deployed and introduce a new computing paradigm, in-network compute. Thus, datacenter systems require special design and implementation to take full advantage of the deployed hardware. The single administrative domain under which each datacenter runs allows breaking backwards compatibility and introducing new protocols to achieve the above goal. This freedom significantly opens up the design space for each problem leading to datacenter-specific and customized solutions and has been a fundamental element of my research so far.

I have been following a very specific methodology in the way I choose and approach research problems that is also reflected in my track record. The first step of this methodology is **measuring and understanding** the target problem, which might require new tools and methodologies. For example, Lancet [9] is such a tool for µs-scale experiments. The second step goes after **improving existing abstractions and systems** either through a better design or better implementation targeting the datacenter. For example, ZygOS [13] improved the existing state-of-the-art at the time in datacenter operating systems for latency critical applications through a better scheduling mechanism. Finally, the third step is to identify the fundamental limitations of existing abstractions and **design new abstractions and protocols** that simplify the implementation of datacenter systems, leverage emerging hardware, and lead to better performance. For example, R2P2 [10] is a new transport protocol specifically designed for datacenter *Remote Procedure Calls (RPCs)* that enables in-network policy enforcement through the use of programmable switches.

Looking ahead I would like to continue my research on datacenter systems aiming to provide all the necessary building blocks to build future cloud and datacenter applications. Specifically, I am interested in answering some of the following questions. *How can we unify the performance metrics used by the operating systems and networking communities, namely latency and flow completion time, and co-schedule the underlying resources? What would be the performance of datacenter systems and how could we build equally performing datacenter systems if we challenge the assumption of the datacenter being a trusted and isolated environment? Do we need programming language support to build tail-tolerant systems in a µs-scale given that existing operating system mechanisms start hitting certain hardware limitations and how would this support look like? How do future datacenter applications look like and what are their basic building blocks and their programming model?*

For the rest of this statement I will briefly describe the research I did during my PhD and how this influenced the community and then I will talk about my future research plans in more detail.

Latency-critical and Tail-tolerant Datacenter Systems

Measuring Tools and Evaluation Methodologies

A major part of my work so far has been focused on understanding tail latency and its sources in latency-critical datacenter systems, such as key-value stores and in-memory databases. To support this effort, I designed tools as well as measuring and evaluation methodologies that are now used either in industry or in subsequent academic efforts adopted by other research groups.

I built LANCET [9], a self-correcting tool designed to measure the open-loop tail latency of µs-scale datacenter applications with high fan-in connection patterns. LANCET is self-correcting as it relies on online statistical tests to determine situations in which tail latency cannot be accurately measured from a statistical perspective. When available, LANCET leverages NIC-based hardware timestamping to measure RPC end-to-end latency. LANCET is an open-source project [3] and since its release has been used and contributed by people from academia and industry, *e.g.,* Amazon's AWS. LANCET's contribution apart from the tool itself and the measuring methodology, is the careful study of factors that can affect latency measurements which is transferable across systems and tools.

The other contribution of my work in this field is the introduction of a systematic evaluation methodology that depends on synthetic microbenchamarks and system modelling with queueing theory. The use of synthetic microbenmarks with configurable service times and request/reply sizes enables matching the theoretic queueing results with the actual system implementation to accurately quantify the system and communication overheads. Our methodology identifies how far is the system implementation from the theoretically optimal one and attributes the sources of inefficiency to communication or system overheads. This methodology was introduced with ZygOS [13] and later used widely by researchers working on scheduling for µs RPCs.

Improving Existing Abstractions

TCP has emerged as the main transport protocol for latency-sensitive, intra-datacenter RPCs running on commodity hardware, as its reliable stream semantics provide a convenient abstraction to build upon. My work approaches systems running on TCP both from an operating systems and a networking perspective.

ZygOS [13] is a system optimized for µs-scale, in-memory computing on multicore servers. It implements a work-conserving scheduler within a specialized operating system designed for high request rates and a large number of network connections. It uses a combination of shared-memory data structures, multi-queue NICs, and inter-processor interrupts to rebalance work across cores. ZygOS was the first system to tackle the topic of scheduling µs-scale RPCs based on queueing theory analysis, while the existing state-of-the-art systems, *e.g.,* dataplane operating systems, focused purely on eliminating system overheads, and does so while still implementing fully-fledged vanilla TCP. After ZygOS several other systems have been proposed to follow up on ZygOS and use ZygOS as a baseline, e.g. Shenango [11] and Shinjuku [4].

From a networking perspective my work extends TCP in two main ways. In [5] I introduce the notion of an SLO-aware flow control mechanism and I implement it on top of TCP by repurposing the existing flow control. The goal of this TCP extension is to advocate for tail-tolerance as a core systems design principle and implements flow control aiming to reduce SLO violations as opposed to avoiding memory exhaustion. Also, I proposed a new scheme for layer 4 load balancing with CRAB [8]. *Connection Redirect Load Balancing* leverages the load balancer only during the TCP connection establishment and bypasses it on the datapath. CRAB requires minimal changes to the TCP stack, it targets internal cloud workloads, and depends on the cloud deployment model for easy adoption. While working on CRAB I had to go through and compare almost all available commodity technologies for building and modifying a networking stack, ranging from direct kernel modifications and Netfilter modules to eBPF, kernel-bypassing and P4, that enabled me to have a hands-one experience and a rather complete picture on how to build and design a networked system using the right technology.

New Abstractions for RPCs

My work on TCP, though, revealed the inefficiencies and the mismatch between TCP's bytestream abstraction and the message oriented nature of RPCs and worked as a motivation for my follow-up work. Identifying the need for efficient µs-scale RPCs in the datacenter I proposed R2P2 [10]. R2P2 is a UDP-based transport protocol specifically designed for RPCs inside a datacenter that exposes the RPC abstraction to the endpoints and the network, making RPCs first-class datacenter citizens. R2P2 is specifically designed for efficient in-network policy enforcement by separating the policy enforcing mechanism from request and reply streaming. Any RPC policy logic can be implemented either in a software middlebox or a programmable switch, *e.g.,* Tofino. By exposing the right abstraction and making the network RPC-aware R2P2 is the catalyst that enables pushing server-side functionality that is common among applications in the transport layer and the network. The work on R2P2 is a first step towards unifying the research on latency-critical systems by network and operating systems communities that have been focusing on different metrics, flow completion time vs RPC latency.

The design of R2P2 enabled the implementation of several RPCs policies and the enrichment of the transport layer with functionality that had to be implemented within each application up until now. The *Join-Bounded-Shortest-Queue* scheduling policy [10] implemented on a programmable middlebox introduced a new compute design in which cores are free of any scheduling decision and just run incoming requests to completion. All the scheduling logic is offloaded to network middleboxes that can schedule RPCs to each core individually. SVEN [7] showcases the flexibility of the new design and the ease of introducing new in-network RPC policies by implementing an SLO-aware flow control mechanism, similarly to [5], leveraging the timestamping functionality of programmable P4 switches. Finally, my work on HovercRaft [6], that received the Best Student Paper Award at Eurosys2020, proposes a new way to build generic fault-tolerant RPC services by providing fault-tolerance at the RPC layer, leveraging the exposed RPC semantics offered by R2P2. HovercRaft deals with the replication trade-off according to which adding nodes to a system can either increase performance at the expense of consistency, or increase resiliency at the expense of performance. HovercRaft follows a systematic approach in identifying potential bottlenecks in state-machine replication and proposes solutions that leverage modern datacenter technology to go after each one of them.

Future Directions

I plan to continue working on µs-scale datacenter systems at the intersection of different computing disciplines aiming to provide more system-level guarantees on top of the tail-latency ones and enable engineers build scalable applications they can reason about using emerging hardware.

Latency vs Flow Completion Time

There is a significant amount of work from the networking and the operating systems communities that focuses on datacenter systems. Those communities, though, approach the problem using different metrics, flow completion time vs RPC latency. My work on R2P2 was a step towards bringing those two communities closer by using network mechanisms and in-network compute to improve RPC scheduling. Exposing the RPC semantics to the network has proven to be beneficial when the bottleneck is on the CPU. I plan to extend the R2P2 work and leverage the RPC abstraction in cases where the network is the bottleneck. Leveraging information about the message size, message type (request/response), RPC priority, *etc.* along with emerging in-network programmability can aid the design of congestion control schemes and traffic engineering approaches. Some initial results towards this direction can be found in the master thesis of one of my students [12], in which we explored a potential design for a congestion control scheme for R2P2 that takes advantage of the R2P2 design. In regards to traffic engineering I would like to revisit the widespread use of ECMP, which implements random scheduling similar to what dataplanes like IX performed before ZygOS, and try to incorporate the benefits of a single queue design based on programmable switches and the information that R2P2 exposes. The ultimate goal in this direction is to be able to co-schedule the network and CPU resources from the end-points in order to achieve better end-to-end RPC latency, instead of blindly optimising for flow completion time and doing so at the scale of a datacenter in µs.

Looking Up the Stack

My work so far has been focusing on operating systems and network mechanisms and remained completely agnostic to the application layer and the programming languages used in this layer. However, as a community we have hit certain hardware limits in trying to build tail-tolerant systems using this approach. ZygOS proposes the optimal solution for µs applications with slight variability while Shinjuku [4] does the same for applications with high variability. Both systems, though, have to resort to relatively heavy mechanisms that leverage inter-processor interrupts in order to achieve their goal and remain application agnostic. I believe it is the right time to start involving the programmer in the quest towards tail-tolerance and predictability. We need programming language abstractions and compiler mechanisms that enable people to write code with tail-tolerance guarantees. Having such support from programming languages can significantly simplify the underlying operating system mechanisms leading to better efficiency. A step towards this direction is the work done by one of my students in semester project [16] in which we extend Clang with automatic coroutine scheduling capabilities. Also, one of the projects I am working on at Microsoft Research is Verona [15], which is a new safe systems programming language with a very interesting concurrency model. My current work on Verona aims at creating a language that offers both safety and tail-tolerance and requires minimal system support, thus it can run on a simplistic but extremely efficient dataplane OS, such as IX, while providing the same guarantees as ZygOS and Shinjuku.

Distrusting the Datacenter

Both R2P2 (non-encrypted communication) and HovercRaft (non-byzantine agreement) depend on the assumption that the datacenter is a trusted environment and the datacenter provider and cloud tenants are not malicious. Breaking this assumption, especially in the face of new exploits, poses a new set of challenges for the design of µs-scale datacenter systems. Extending RPC transport protocols, *e.g.,* R2P2, with support for encrypted RPCs without losing the existing performance benefits is challenging and might require significant redesign. Doing so on top of a 400G network will require several changes across all the software layers and the hardware infrastructure. A first step towards this direction can be found in one of my students semester project where we explore use of TLS 1.3 on top of R2P2 [1]. In the same direction, I would like to explore how can we apply the ideas proposed in HovercRaft and use of in-network compute for a byzantine agreement scenario. From a compute perspective, I am interested in exploring how can we maintain the tail-tolerance guarantees already provided for datacenter applications when we challenge the confidentiality or isolation guarantees. Can we still run at a µs-scale on top of a potentially compromised infrastructure or using potentially malicious or faulty software libraries? Our work on Enclosure [2] provides language and system support for intra application compartmentalization. Also, at MSR I am working on building latency critical applications inside trusted execution environments (TEEs), which I view as the only way for the integral adoption of cloud computing. My experience with kernel-bypassing for performance seems to be transferable to the world of confidential computing aiming at reducing the TCB. So, building performant but also confidential and secure systems is a direction I would like to actively pursue.

Future Datacenter Systems

My research so far and my immediate research directions can be seen as the necessary building blocks towards the future cloud and datacenter systems. My vision for the future is moving from existing infrastructure-centric to more application-centric designs. Rather than pushing towards disaggregation at the OS-level, I think that disaggregation should happen through disaggregated services and serverless computing. I believe there should be a clear separation between stateful services that run on compute-capable data-stores and stateless services that can run anywhere. Serverless is still a millisecond-scale application, mainly because of system bottlenecks that we are trying to push, *e.g.,* our work on REAP optimises cold boot times for serverless workloads [14]. My goal is to push serverless to the µs-scale, which will unlock the way we design and build efficient datacenter systems due to the fine granularity of computation. The programming model for this kind of future datacenter systems is not clear. In the long run I want to explore what is the ideal programming model, the necessary abstractions, the underlying programming language mechanisms, and the OS and network functionality, to design and build future cloud and datacenter systems that can provide tail-tolerance, fault-tolerance, and confidentiality guarantees.

References

[1] Encrypted R2P2. `https://marioskogias.github.io/students/solignac.pdf`.

[2] Adrien Ghosn, Marios Kogias, James Larus, and Ed Bugnion. Enclosure: language-based restriction of untrusted libraries. In *26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII)*, 2021.

[3] Lancet source code. `https://github.com/epfl-dcsl/lancet-tool`.

[4] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. Shinjuku: Preemptive Scheduling for µsecond-scale Tail Latency. In *Proceedings of the 16th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 345–360, 2019.

[5] Marios Kogias and Edouard Bugnion. Flow control for Latency-Critical RPCs. In *Proceedings of the 2018 Workshop on Kernel-Bypass Networks (KBNETS@SIGCOMM)*, pages 15–21, 2018.

[6] Marios Kogias and Edouard Bugnion. HovercRaft: achieving scalability and fault-tolerance for microsecond-scale datacenter services. In *Proceedings of the 2020 EuroSys Conference*, pages 25:1–25:17, 2020.

[7] Marios Kogias and Edouard Bugnion. Tail-tolerance as a systems principle not a metric. In *4th Asia-Pacific Workshop on Networking*, APNet '20, page 16–22, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Marios Kogias, Rishabh Iyer, and Edouard Bugnion. Bypassing the load balancer without regrets. In *Proceedings of the 2020 ACM Symposium on Cloud Computing (SOCC)*, pages 193–207, 2020.

[9] Marios Kogias, Stephen Mallon, and Edouard Bugnion. Lancet: A self-correcting Latency Measuring Tool. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, pages 881–896, 2019.

[10] Marios Kogias, George Prekas, Adrien Ghosn, Jonas Fietz, and Edouard Bugnion. R2P2: Making RPCs first-class datacenter citizens. In *Proceedings of the 2019 USENIX Annual Technical Conference (ATC)*, pages 863–880, 2019.

[11] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving High CPU Efficiency for Latency-sensitive Datacenter Workloads. In *Proceedings of the 16th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 361–378, 2019.

[12] Konstantinos Prasopoulos. Extending r2p2 with congestion control and request-level scheduling. http://infoscience.epfl.ch/record/277844, 2020.

[13] George Prekas, Marios Kogias, and Edouard Bugnion. ZygOS: Achieving Low Tail Latency for Microsecond-scale Networked Tasks. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, pages 325–341, 2017.

[14] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Ed Bugnion, and Boris Grot. REAP: Record-and-Prefetch Serverless Functions Orchestration. In *26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII)*, 2021.

[15] Verona Programming Language. `https://www.microsoft.com/en-us/research/project/project-verona/`.

[16] Introducing scheduling decisions in cooperative multitasking: an automated approach. `https://marioskogias.github.io/students/weber.pdf`.