Coalescent Computing

Kyle C. Hale
khale@cs.iit.edu
Illinois Institute of Technology
Chicago, Illinois, USA

ABSTRACT

As computational infrastructure extends to the edge, it will increasingly offer the same fine-grained resource provisioning mechanisms used in large-scale cloud datacenters, and advances in low-latency, wireless networking technology will allow service providers to blur the distinction between local and remote resources for commodity computing. From the users' perspectives, their devices will no longer have fixed computational power, but rather will appear to have flexible computational capabilities that vary subject to the shared, disaggregated edge resources available in their physical proximity. System software will transparently leverage these ephemeral resources to provide a better end-user experience. We discuss key systems challenges to enabling such tightly-coupled, disaggregated, and ephemeral infrastructure provisioning, advocate for more research in the area, and outline possible paths forward.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Hardware** → *Wireless devices*; • **Software and its engineering** → **Cloud computing**.

KEYWORDS

edge computing, disaggregated hardware, operating systems, wireless networks

ACM Reference Format:
Kyle C. Hale. 2021. Coalescent Computing. In *12th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '21), August 24–25, 2021, Hong Kong, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3476886.3477503

APSys '21, August 24–25, 2021, Hong Kong, China

ACM ISBN 978-1-4503-8698-2/21/08…$15.00
https://doi.org/10.1145/3476886.3477503

1 INTRODUCTION

We envision edge deployments (i.e., "cloudlets" [52]) that expose virtualized resources which can *transparently* augment user devices, and which can automatically scale up or down based on available resources, user demand, user proximity (network latency), available network bandwidth, and spot pricing. From the users' perspective, it appears as if their device (laptop, thin client, or smart phone) acquires increased computational power (increased memory or disk capacity, increased CPU count, or high-end GPU) when they wander near such a deployment, for example, into their local coffee shop[1]. When the user leaves the area, the resources are revoked, and the user's device appears as it did before.

Since Satyanarayanan first laid out the basis for this vision nearly two decades ago with Cyber Foraging [5], hardware, software, and networking technologies have advanced to the point where it will soon be possible for the transparent *coalescence* of disaggregated computational resources to client machines to occur at a fine granularity and over short time scales. We call this notion *Coalescent Computing*, a type of Cyber Foraging for disaggregated hardware at the edge.

Cyber Foraging generally relies on discoverable services in users' local environments, and on the ability to *offload* application components to remote—and generally more capable—machines [51]. This offloading usually happens at the granularity of virtual machines [24]; while the end user may be unaware of the local/remote distinction in this scenario, it is still present for the application programmer and system software. Though applications can be automatically partitioned into loosely-coupled components to make them amenable to such cloud offload [11, 29], and while VM migration can be used to ship applications to the cloud transparently [23], we believe that there is an opportunity to leverage the increasingly hierarchical and disaggregated structure of cloud resources at the edge [60, 61] to support applications that are more tightly coupled, including enhanced gaming, augmented reality (AR) [70], virtual reality (VR) [65], interactive data analysis [53], and IoT [68].

One goal of classic distributed operating system work from the 70s and 80s was to hide loosely-coupled distributed machines behind the illusion of a single, logical system (a

[1]The authors acknowledge that since we are currently living in a pandemic some readers might find this particular example far-fetched!

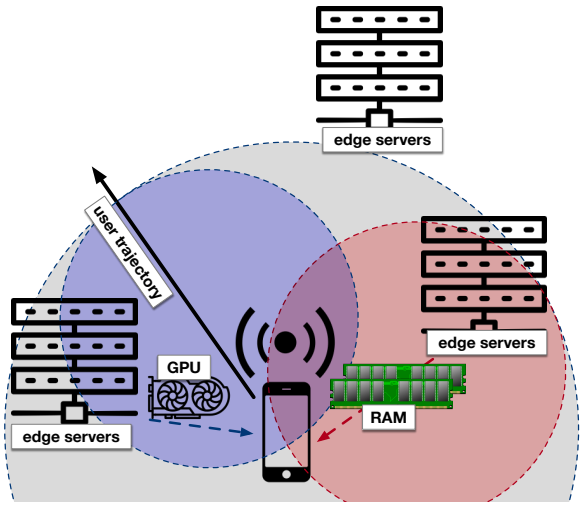

Figure 1: Coalescent Computing

.

single system image [9]). While commercially this did not come to pass (except for limited components, e.g. file systems), a natural question now arises: is it time to reconsider this aspiration for our modern computational ecosystem? In the datacenter, the answer seems to be *yes*. High-speed interconnects (e.g. InfiniBand) within the datacenter have increasingly made resource sharing between systems feasible [31, 62], for example shared remote memory [1, 15, 45, 71] or remote swap [2, 22, 49]. I/O device virtualization is becoming more sophisticated, with efficient offload enabled by API remoting [4, 16, 50], and device sharing among tenants [66]. These trends, along with hardware proposals for scale-out systems [21, 36, 43] and disaggregated hardware on the horizon [3, 12, 18, 28] point towards a datacenter that consists of loosely-coupled, disaggregated resources. LegoOS, a notable first step in disaggregated operating systems, embraces this view of the datacenter while retaining Linux ABI compatibility [55], and GiantVM demonstrates how to virtually and transparently compose datacenter resources [69].

While datacenters obviously benefit from low-latency, wired interconnects and relatively static hardware configurations, the momentum in disaggregation is encouraging for commodity computing at the edge as well, and we argue that there is a ripe opportunity for OS research to make Coalescent Computing a reality. Below we describe Coalescent Computing in more detail, discuss some of its key research challenges, and ideas for OS design to investigate the space.

2 COALESCENT COMPUTING

Coalescent Computing can be captured succinctly with the following principle:

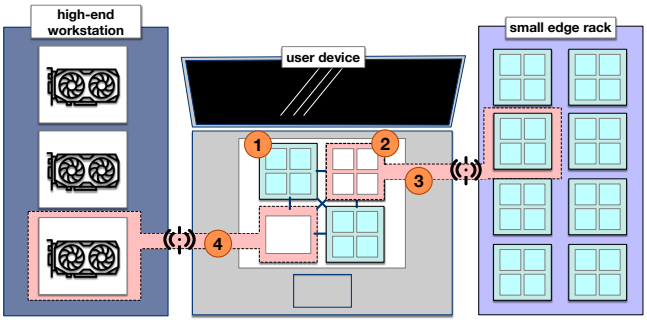

Figure 2: Coalescence of disaggregated remote cores and a GPU in the user's proximity.

> **Coalescence Principle:** Users' devices experience a coalescence of resources proportional to proximity as users move through the physical environment.

Figure 1 depicts the Coalescence Principle at work. The number of resources coalesced into a user's device is inversely proportional to the user's network distance to the hosting machines[2]. While in most cases those hosting machines will be stationary, in some cases, users with less constrained devices may offer their resources to nearby users, as in Femtocloud [25]. Other metrics, such as pricing, network congestion, and power budgets for the systems hosting disaggregated resources will also affect availability.

As a user navigates the physical environment, the OS on their device queries nearby resource availability, and puts in bids for resources based on current and historical system load. For instance, a user that just finished a recorded Zoom call might cause a CPU spike that triggers their OS to bid for nearby leasable CPUs to aid in video encoding. If no such resources are available, the system can either borrow resources from the traditional cloud (with a latency penalty), or fall back to local resources. Figure 2 depicts a scenario where a user is playing a CPU and GPU-intensive game that outstrips the abilities of his or her laptop. The user is in nearby proximity of a high-end workstation housing several GPUs and an edge rack that contains a collection of disaggregated CPUs. The user sees both physical resources on the local machine (1) and virtual resources from remote systems (2). To accommodate system load, the device's Coalescent OS transparently discovers, negotiates, and acquires a virtual GPU 4 and four virtual cores 3 from the rack over the wireless link. This reactionary resource provisioning is reminiscent of computational sprinting [38, 48] and JIT-provisioned Cyber Foraging [24], but it happens at the granularity of disaggregated cores and devices. Note that a corollary of the Coalescence Principle is that as users leave

[2]This may or may not correspond to physical distance, e.g. in wired settings.

the environment resources are relinquished and the system gracefully migrates any necessary computations or state back to the local machine, or to the cloud if WAN latencies can be tolerated.

We expect a system implementing Coalescent Computing to have the following properties:

Transparency. While users may set coalescence policies ahead of time, the system transparently acquires and relinquishes resources nearby in the environment; this is a key distinguisher from typical cloud offload. Acquisition here does not mean sole ownership of the physical remote resource, and does not necessarily imply that the user should be charged for it. For example, an idling user's system may make resource reservations, but as long as the user is idle, the OS or monitor on the remote system is free to schedule other work. Quality-of-Service (QoS) policies and the degree of sharing can be determined by providers, and will likely change from user to user. Because users' resource acquisition policies may be at odds with one another, and because applications have diverse requirements, policy enforcement will involve solving a challenging, multi-objective optimization problem [20] on hosts that expose resources. While there are effective techniques for solving such scheduling problems in the datacenter—for example with recommender systems [13] or reinforcement learning and Bayesian optimization [47]—in this setting user mobility will significantly affect resource availability and users' devices may coalesce resources from different providers, rendering a centralized scheduling mechanism ineffective. One possible path forward is to combine application profiling (informing resource *requests*) with decentralized versions of ML-based schedulers (guiding resource *grants*). Thus, sets of leasable resources (servers, desktops, and possibly user devices) form ad hoc networks to run a distributed, coalescent scheduler.

The user can monitor currently "attached" resources using familiar means. For example, `/proc/cpuinfo` in a Coalescent OS exposing a Linux-like interface would include both physical CPUs on the device and virtual CPUs coalesced from a nearby edge server. Similarly, `/proc/meminfo` would show remotely coalesced pages (though sub-page granularity remote memory is a possibility [49]).

At a surface level, the OS sees the remote CPUs (and other resources) just like normal CPUs, and once they are properly initialized and booted, the OS can schedule work on them. However, the OS must take care in how it schedules work on remote resources when applications are tightly coupled, so must have some notion of resource localization (see Section 3.2). This bears some similarity to NUMA-awareness, but is more challenging given the inherent dynamism of resources whose coalescence depends on user proximity.

Performance. Users will expect their devices to be responsive. While there is more flexibility here than in the datacenter environment, the underlying technology presents more challenges too (Section 3). In particular, as resources attach and detach from user devices, it should not perceptibly affect response times.

While the single-system image abstraction is a compelling one (e.g., CPU cores come and go as the user moves around), not all applications will want to *use* those cores, since their use comes with a latency penalty. The system must be aware of the distinction between latency-sensitive and throughput-sensitive workloads [56], and must guide resource coalescence with that in mind. We believe that there is likely a sweet spot for applications that thrive in a coalescent setting. For applications with components that communicate quite often (e.g., a tightly-coupled, multi-threaded stencil code), decoupling the components will incur a significant penalty. On the other hand, loosely-coupled applications that perform bulk computations (e.g., rendering a single scene) can be sufficiently handled by offloading to a distant cloud. Thus, identifying applications that fit into this sweet spot is a primary concern.

Resilience. Though we can envision Coalescent Computing extending to wired environments[3], systems will more often need to make do with unreliable wireless connections. A Coalescent OS must deal with dropped connections gracefully, for example using replication and fail-over, or by periodic checkpointing. In any case, techniques applied to achieve resilience should avoid centralized coordination given the ephemeral proximity of resources. However, some systems—for example edge servers housed in a back room cabinet—will be more static by nature, will have a constant power source, and will likely have a reliable wired connection to the internet, and thus should be weighted more heavily when choosing coordinating nodes.

Customizability. While users need not normally tend to resource coalescence policies, we anticipate that there will arise scenarios where customization will be advantageous. For example, users may set a lower threshold on battery levels at which the system discovers, negotiates, and leases resources, thus limiting power consumption by the wireless radio and by the system itself. Even if one user has a high-end laptop, he or she likely would not want another user pegging one of the CPUs when the battery is on its last leg. Other users might prefer more detailed performance tuning, for example setting thresholds on swap space using remote memory, capacity limits on leased resources, CPU load thresholds for offloading, and so on.

[3]For example, inductive charging surfaces seen in some coffee shops now might one day incorporate network interfaces.

Technology	Latency
SoL lower bound at 10m	33 ns
Cross-core cache-coherence	100-200 ns [30]
soNUMA (proposed)	300 ns [43]
Cross-socket (QPI)	355 ns [10]
Inter-processor Interrupts (IPI)	~500 ns [26]
PCIe Gen 3	900 ns [40]
InfiniBand RDMA (one-sided)	1 μs [32]
WiFi 6E (reported)	~2 ms [14]
5G URLLC (reported)	1 ms [19, 37]
5G (first-hop)	14 ms [39]
Typical WiFi (90^{th} %-ile)	20 ms [58]

Table 1: One-way latency for common and emerging networking technologies.

Privacy and Security. When users offload computation to cloud resources or instantiate VMs on public infrastructure, they place some degree of trust in the provider, since they direct the action. With Coalescent Computing, a user's application may be run on untrusted hardware, potentially divulging sensitive information. Some malicious users may be incentivized to lease out their resources just to compromise other users' data. Others may coalesce resources from nearby users (possibly coordinating with other bad actors nearby) to carry out a denial-of-service attack. Systems must have mechanisms in place to mitigate such scenarios. This is a problem that also plagues decentralized volunteer computing systems [17, 33]. Proper isolation using hardware support, virtualization, and collaborative monitoring and reporting of bad actors can alleviate the effects of malicious behavior.

3 CHALLENGES

We now discuss major challenges both in hardware and in OS design that impede progress in realizing Coalescent Computing.

3.1 Hardware

The overriding challenge for Coalescent Computing from the hardware perspective will be the performance characteristics of wireless links. Table 1 lists single-hop latencies for various interconnects up and down the stack reported by others. Cache line transfers on the coherence network between Nehalem cores land in the 100-200ns range, whereas high-performance InfiniBand cards are still more than 3X that latency at ~1μs. As Shan et al. have already shown in the datacenter environment, this puts coherent resources off the table for now [55]. This especially rings true for wireless technologies (last four rows of Table 1). Typical WiFi connections have reasonably low first-hop latency at 20ms, but there is a long tail that puts the damper on deterministic performance. However, emerging, ultra-low latency wireless standards like 5G URLCC (designed with applications like wireless factory automation and AR in mind) and WiFi 6E bring the latency down by an order of magnitude and are reported to reduce latency variance significantly. Coherence will still be out of reach, but with the right OS support we believe Coalescent Computing can be realized over these low-latency wireless links. For reference, the first row of the table shows the speed-of-light delay at 10m, which we can view as a lower bound on the latency of future wireless networking between edge systems. While there is much work on characterizing and improving the performance of wireless links, little has been done to guide automated decisions based on their properties. In particular, for a coalescent system to work properly, it must be able to infer signal strength (and user distance) accurately in order to project the impacts on application performance and thus guide coalescence dynamically. This is an open problem.

Unfortunately, current wireless interfaces are not suitable for operating with disaggregated resources. Prototype systems for disaggregated hardware today make heavy use of RDMA capabilities and fixed network latencies. WiFi interfaces could be optimized for Coalescent Computing, for example by customizing the wire protocol for resource acquisition, and by integrating low-power mechanisms for resource discovery, as in BlueTooth Low Energy (BLE). These NICs might also incorporate features we see in high-end cards today like RDMA, atomics, and memory protection. The NICs might also be integrated near the processors to act as a proxy socket to facilitate communication between remote resources, as in soNUMA [43].

While the OS may employ loosely-coupled monitors on remote resources (Section 3.2), users may want to customize the software they run on these resources. This will require enhanced lightweight virtualization in wireless NICs, namely self virtualization (e.g., SR-IOV) and boot protocols that incorporate disaggregated hardware (extended PXE). NICs on the users' systems must coordinate with the BIOS (e.g. via a lightweight platform management controller or a BMC) in order to keep hardware information exposed to the OS (namely, ACPI tables that enumerate NUMA regions and processor information like the SRAT and SLIT tables) consistent with coalesced resources. ACPI likely needs to be extended to support Coalescent Computing, and platform hardware will need to route the boot sequence (e.g. the SIPI and IPI sequence on x86 chips) through something like an APICv [41] rather than applying the traditional *trap-and-emulate* model.

3.2 Software

A Coalescent OS will need to support the following: performance, disaggregation, resource discovery, adaptation, hardware heterogeneity, and fault tolerance. Several OSes from the research community support some of these features, but not all. For example, LegoOS is the first OS designed for disaggregated hardware [55], and provides a good foundation to build upon for Coalescent Computing. The idea of stateless, loosely-coupled monitors running on disaggregated hardware components will serve a Coalescent OS as well. However, the LegoOS design focuses on datacenter applications, and the ExCache-based memory management, the global resource managers, and the InfiniBand/RDMA-based RPC will not transfer easily to a wireless edge setting without significant hardware enhancements.

Performance. To reconcile privacy and performance, users will likely want their code and data to reside in isolated environments. This means that monitors will need to employ very lightweight, fast-start hardware virtualization, which we have previously shown is possible on the order of microseconds[4]. Light-weight, virtual execution environments will be launched on-demand to host second-level monitors from the mobile user's system. Hardware monitors will isolate user monitors from one another. Virtualization hardware enhancements discussed in the previous section will make this more efficient, and a Coalescent OS will likely incorporate something like the *boot drivers* used in Barrelfish/DC to account for dynamically changing CPU information not supported in ACPI [67]. The CPU boot process will look much more like the plug-and-play PCI probing process present in commodity OSes today. For undersubscribed CPUs, the resource monitor may use CPU hot-remove functionality to space-partition the user monitor, reminiscent of co-kernels in Pisces [46]. As with Barrelfish/DC, decoupling the OS from the underlying hardware will allow for greater flexibility with dynamic OS updates as well, as was also demonstrated in K42 [8].

Performance will be mainly limited by network latency and bandwidth. A Coalescent OS will have to employ aggressive techniques to hide network latency and variability. The OS can avoid expensive coherence traffic by using message passing in lieu of shared memory, for example as is done in Barrelfish [7] and LegoOS. Serialization costs and software overheads must also be avoided, as we are learning with disks as SSDs become faster [35]. For remote memory performance, skewed access distributions may help [21], allowing caches to be used to take advantage of temporal locality, but it is unlikely to produce the same benefits we see in the datacenter. That said, similarities between users in geographical proximity may offer hope, and the same principles that enable CDNs will present opportunities for deduplication and sharing in edge systems [61].

Coalescent systems will benefit from QoS policies. These policies can be set based on provider inputs (e.g. informed by user account balance), social credits ("how many CPU hours has the user leased out?"), current system load, physical proximity, and the user's affinity for particular resources ("only coalesce memory, not CPU or accelerators").

The system should also employ best effort coalescence of resources; namely, if network conditions are incapable of providing adequate performance, resource negotiations should fail, and applications can run on local resources or fall back to the traditional cloud. This best-effort behavior has already been demonstrated for servicing I/O requests in MittOS [27].

Generally speaking, as Schwarzkopf et al. aptly point out [54], deterministic performance was the albatross for early distributed OSes, and we must be mindful of the lessons learned there [59, 63]. Hardware improvements will certainly help, but exposing performance variability to the OS is paramount for it to make acceptable decisions.

Heterogeneity. Disaggregated CPUs, GPUs, FPGAs, memory, and storage will inevitably be more heterogeneous than in a typical datacenter. A Coalescent OS must handle this heterogeneity transparently. Monitors written for different devices can expose a unified interface, but applications must be able to run on diverse hardware, including different ISAs, especially as competitors to x86 gain prominence. This system might require applications be compiled into fat binaries, but a more flexible approach would employ an intermediate representation (IR) to dynamically adapt the application to the ISAs of nearby resources, as in Helios [42]. Such a system would make judicious use of just-in-time (JIT) compilers on edge nodes, or in cases where performance is less critical, language VMs. Using JIT compilation to address heterogeneity adds another layer of complexity for performance, as it can introduce even more variability. Managing this variability is critical for Coalescent Computing, but we have only scratched the surface of minimizing JIT compilation latency [34].

Resource Discovery. As users navigate the physical environment, their devices must query nearby systems for available resources. This resource discovery process must occur often enough to react to load spikes, but not so often as to drain device battery and congest the local network. The OS and hardware might employ UPnP here [44], as in Slingshot [57], but the protocol will likely need to be enhanced to include resource load information and performance characteristics. When negotiating coalescence, the OS will automatically

[4]Citation elided for double-blind review

choose a subset of nearby resources subject to user preferences and system load.

Programming Model. The Coalescent OS can by default transparently migrate computation between local and remote machines. For example, for each new vCPU added to the system via Coalescence, the OS exposes a new run queue, e.g., over distributed shared memory. The OS can add a thread to the remote run queue as it would on the local system, but with a performance penalty. For example, the code in Listing 1 shows a trivial example of creating a worker thread that processes tasks in a shared queue. The Coalescent OS is free to schedule this thread on a remote vCPU if available. However, since the user is mobile, that remote vCPU may disappear. The remote worker thread might dequeue work from the shared queue then fail, losing that work. There are of course many techniques for handling failures and ensuring consistency in distributed systems, but in a language like C where mutations on shared state can happen anywhere, it is quite challenging to apply these techniques transparently.

```
static void worker() {
    while (1) {
        work_t * work = dequeue_work();
        do_work(work);
    }
}

void main() {
    pthread_t thr;
    pthread_create(&thr, NULL, worker, NULL);
    pthread_join(thr, NULL);
}
```

Listing 1: Creating a worker thread.

One possibility is to expose remote resources and the potential for failure to the programmer. Listing 2 shows such an example using a CC-aware wrapper around the pthreads runtime. Here the programmer explicitly places constraints on the remote vCPUs that the thread can run on by specifying the maximum acceptable latency to the remote vCPU in μs. The programmer also indicates that the system should favor remote vCPUs over local ones when available (`EAGER_REMOTE`). Finally, the programmer specifies that when a failure is detected by the runtime, the thread should be recreated on the local machine after a failure is handled by user-specified code (here the programmer provides code to recover the work queue, e.g., with a persistent write-ahead log).

```
#include <cthread.h>
static struct cthread_attrs {
    .latency_bound = 100, // usec
    .strategy      = EAGER_REMOTE,
    .failure       = RETRY_LOCAL,
} cta;

static int handle_failure() {
    return recover_work_queue();
}

void main() {
    cthread_t thr;
    cthread_create(&thr, &cta, fun, NULL,
    handle_failure);
    cthread_join(thr, NULL);
}
```

Listing 2: Creating a worker thread with failure recovery using an explicit CC API.

In many cases, it will be preferable to manage *functions* running on remote resources, rather than execution contexts. In this case, the Coalescent OS can expose a function-as-a-service (FaaS) API as well. In addition to the typical FaaS event-triggered function invocation model, a CC FaaS API might also allow for RPC-like, *synchronous* invocations via language annotations. For example, a programmer might specify that a function can run on remote resources by using a *virtine* (virtual subroutine) [64], as shown in Listing 3.

```
virtine int fun() {
    do_work();
}
```

Listing 3: Coalescence with a virtine.

In this case, if remote resources are available, the invocation of `fun` will cause a light-weight, isolated VM (or container) to be spawned on the remote vCPU and the function will run to completion.

Adaptation and Fault Tolerance. A Coalescent OS will need to adapt to changing network conditions and resource availability. Ideas from systems like Chroma apply here [6]; resources must be monitored, and application usage estimated so that applications can scale up and down depending on what is available. The system may leverage redundant computations across multiple resources to mitigate tail latency and for resilience.

To handle failures, a coalescent system will likely employ replicas and periodic checkpointing. Replication will be more challenging than in the datacenter environment given increased mobility, but replica selection can be informed by mobility characteristics of different systems. A server plugged into the wall would be a better choice for fail-over rather than a nearby laptop. Replicas might exist in hierarchies based on the environment. For example, a secondary

replica may be placed on the nearby edge server, and a tertiary replica may be instantiated in the cloud with relaxed consistency. Append-only storage can be used to persist state changes and aid in failure recovery.

When component or connection failures occur, or when the user moves out of range , the system must decide how to react. This will largely depend on application resource demands and latency sensitivity. For example, when a user training a neural network decides to move away from a resource-rich area, the training can be shipped off to the cloud to complete. However, a user running an immersive augmented reality application may prefer to have all computation and state migrated back to the local device, perhaps trading off degraded quality for responsiveness.

4 CONCLUSION

Several challenges remain for Coalescent Computing which we do not touch on here, but which we do plan to investigate. These include the storage interface, resource naming, and a more detailed treatment of privacy and security (e.g. authentication).

The building blocks for Coalescent Computing are gradually being put in place. We will soon stand at the confluence of disaggregated hardware, hierarchically distributed clouds, and ultra low-latency wireless networks. We argue that exploring systems that support this model will not only put more computational power at users' fingertips, but will also shed light on new avenues of systems research.

ACKNOWLEDGEMENTS

This paper would not have been possible without valuable discussions and feedback from Conghao Liu, Brian Tauro, Nicholas Wanninger, Rich Wolski, Peter Dinda, and Nikos Hardavellas. This work is supported by the United States National Science Foundation via awards CNS-1718252, CNS-1763612, CNS-1730689, CCF-1757964, CCF-2029014, and CCF-2028958.

REFERENCES

[1] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. 2018. Remote Regions: A Simple Abstraction for Remote Memory. In *Proceedings of the 2018 USENIX Annual Technical Conference* (Boston, MA, USA) *(USENIX ATC '18)*. USENIX Association, USA, 775–787. https://dl.acm.org/doi/10.5555/3277355.3277430

[2] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can Far Memory Improve Job Throughput?. In *Proceedings of the* 15^{th} *European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 14, 16 pages. https://doi.org/10.1145/3342195.3387522

[3] Krste Asanović. 2014. FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers. In *Proceedings of the* 12^{th} *USENIX Conference on File and Storage Technologies (FAST '14)*. USENIX Association, Santa Clara, CA.

[4] Marco Bacis, Rolando Brondolin, and Marco D. Santambrogio. 2020. BlastFunction: An FPGA-as-a-Service System for Accelerated Serverless Computing. In *Proceedings of the* 23^{rd} *Conference on Design, Automation and Test in Europe* (Grenoble, France) *(DATE '20)*. EDA Consortium, San Jose, CA, USA, 852–857.

[5] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. 2002. The Case for Cyber Foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop* (Saint-Emilion, France) *(EW '10)*. Association for Computing Machinery, New York, NY, USA, 87–92. https://doi.org/10.1145/1133373.1133390

[6] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. 2003. Tactics-Based Remote Execution for Mobile Computing. In *Proceedings of the* 1^{st} *International Conference on Mobile Systems, Applications and Services* (San Francisco, California) *(MobiSys '03)*. Association for Computing Machinery, New York, NY, USA, 273–286. https://doi.org/10.1145/1066116.1066125

[7] Andrew Baumann, Paul Barham, Pierre Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the* 22^{nd} *ACM Symposium on Operating Systems Principles (SOSP '09)*. 29–44.

[8] Andrew Baumann, Gernot Heiser, Jonathan Appavoo, Dilma Da Silva, Orran Krieger, Robert W. Wisniewski, and Jeremy Kerr. 2005. Providing Dynamic Update in an Operating System. In *Proceedings of the 2005 USENIX Annual Technical Conference* (Anaheim, CA) *(USENIX ATC '05)*. USENIX Association, USA, 32.

[9] Rajkumar Buyya, Toni Cortes, and Hai Jin. 2001. Single System Image. *The International Journal of High Performance Computing Applications* 15, 2 (2001), 124–135. https://doi.org/10.1177/109434200101500205

[10] Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2016. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *Proceedings of the* 53^{rd} *Annual Design Automation Conference* (Austin, Texas) *(DAC '16)*. Association for Computing Machinery, New York, NY, USA, Article 109, 6 pages. https://doi.org/10.1145/2897937.2897972

[11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. 2011. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proceedings of the* 6^{th} *Conference on Computer Systems* (Salzburg, Austria) *(EuroSys '11)*. Association for Computing Machinery, New York, NY, USA, 301–314. https://doi.org/10.1145/1966445.1966473

[12] I-Hsin Chung, Bulent Abali, and Paul Crumley. 2018. Towards a Composable Computer System. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region* (Chiyoda, Tokyo, Japan) *(HPC Asia '18)*. Association for Computing Machinery, New York, NY, USA, 137–147. https://doi.org/10.1145/3149457.3149466

[13] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the* 18^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. Association for Computing Machinery, New York, NY, USA, 77–88. https://doi.org/10.1145/2451116.2451125

[14] Gino Dion. 2020. Wi-Fi 6 and Wi-Fi 6E: better, faster, more. https://www.nokia.com/blog/wi-fi-6-and-wi-fi-6e-better-faster-more/. Accessed 2020-01-31.

[15] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *Proceedings of the* 11^{th} *USENIX Symposium on Networked Systems Design and*

Implementation (NSDI '14). USENIX Association, Seattle, WA, 401–414. https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi{ć}

[16] José Duato, Antonio J. Peña, Federico Silla, Rafael Mayo, and Enrique S. Quintana-Ortí. 2010. rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In *Proceedings of the 2010 International Conference on High Performance Computing and Simulation* (Caen, France). 224–231. https://doi.org/10.1109/HPCS.2010.5547126

[17] Arnaud Durand, Mikael Gasparian, Thomas Rouvinez, Imad Aad, Torsten Braun, and Tuan Anh Trinh. 2015. BitWorker, a Decentralized Distributed Computing System Based on BitTorrent. In *Proceedings of the* 13^{th} *International Conference on Wired/Wirless Internet Communications* (Malaga, Spain) *(WWIC '15)*. Springer, 151–164. https://doi.org/10.1007/978-3-319-22572-2_11

[18] Paolo Faraboschi, Kimberly Keeton, Tim Marsland, and Dejan Milojicic. 2015. Beyond Processor-centric Operating Systems. In *Proceedings of the* 15^{th} *Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association, Kartause Ittingen, Switzerland. https://www.usenix.org/conference/hotos15/workshop-program/presentation/faraboschi

[19] Thomas Fehrenbach, Rohit Datta, Bariş Göktepe, Thomas Wirth, and Cornelius Hellge. 2018. URLLC Services in 5G Low Latency Enhancements for LTE. In *Proceedings of the* 88^{th} *IEEE Vehicular Technology Conference (VTC-Fall)*. 1–6. https://doi.org/10.1109/VTCFall.2018.8690663

[20] Yaru Fu, Xiaolong Yang, Peng Yang, K.Y. Wong, Zheng Shi, Hong Wang, and Tony Q.S. Quek. 2021. Energy-efficient offloading and resource allocation for mobile edge computing enabled mission-critical internet-of-things systems. *EURASIP Journal on Wireless Communications and Networking* (Feb. 2021). https://doi.org/10.1186/s13638-021-01905-7

[21] Vasilis Gavrielatos, Antonios Katsarakis, Arpit Joshi, Nicolai Oswald, Boris Grot, and Vijay Nagarajan. 2018. Scale-out ccNUMA: Exploiting Skew with Strongly Consistent Caching. In *Proceedings of the* 13^{th} *EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 21, 15 pages. https://doi.org/10.1145/3190508.3190550

[22] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *Proceedings of the* 14^{th} *USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. USENIX Association, Boston, MA, 649–667. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[23] Kiryong Ha, Yoshihisa Abe, Thomas Eiszler, Zhuo Chen, Wenlu Hu, Brandon Amos, Rohit Upadhyaya, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2017. You Can Teach Elephants to Dance: Agile VM Handoff for Edge Computing. In *Proceedings of the* 2^{nd} *ACM/IEEE Symposium on Edge Computing* (San Jose, California) *(SEC '17)*. Association for Computing Machinery, New York, NY, USA, Article 12. https://doi.org/10.1145/3132211.3134453

[24] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. 2013. Just-in-Time Provisioning for Cyber Foraging. In *Proceeding of the* 11^{th} *Annual International Conference on Mobile Systems, Applications, and Services* (Taipei, Taiwan) *(MobiSys '13)*. Association for Computing Machinery, New York, NY, USA, 153–166. https://doi.org/10.1145/2462456.2464451

[25] Karim Habak, Mostafa Ammar, Khaled A. Harras, and Ellen Zegura. 2015. Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge. In *Proceedings of the* 8^{th} *IEEE International Conference on Cloud Computing* (New York City, NY, USA) *(CLOUD '15)*. IEEE, 9–16. http://doi.org/10.1109/CLOUD.2015.12

[26] Kyle C. Hale and Peter Dinda. 2018. An Evaluation of Asynchronous Events on Modern Hardware. In *Proceedings of the* 26^{th} *IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (Milwaukee, WI, USA) *(MASCOTS '18)*. IEEE. http://doi.org/10.1109/MASCOTS.2018.00041

[27] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. 2017. MittOS: Supporting Millisecond Tail Tolerance with Fast Rejecting SLO-Aware OS Interface. In *Proceedings of the* 26^{th} *Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 168–183. https://doi.org/10.1145/3132747.3132774

[28] Hewlett-Packard, Inc. [n.d.]. The Machine: A New Kind of Computer. https://www.hpl.hp.com/research/systems-research/themachine/. Accessed: 2020-01-10.

[29] Galen C. Hunt and Michael L. Scott. 1999. The Coign Automatic Distributed Partitioning System. In *Proceedings of the* 3^{rd} *Symposium on Operating Systems Design and Implementation* (New Orleans, Louisiana, USA) *(OSDI '99)*. USENIX Association, USA, 187–200.

[30] Stefan Kaestle, Reto Achermann, Roni Haecki, Moritz Hoffmann, Sabela Ramos, and Timothy Roscoe. 2016. Machine-Aware Atomic Broadcast Trees for Multicores. In *Proceedings of the* 12^{th} *USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX Association, Savannah, GA, 33–48. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/kaestle

[31] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *Proceedings of the 2016 USENIX Annual Technical Conference* (Denver, CO, USA) *(USENIX ATC '16)*. USENIX Association, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[32] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16)*. USENIX Association, Denver, CO, 437–450. https://www.usenix.org/conference/atc16/technical-sessions/presentation/kalia

[33] Nils Kopal, Matthäus Wander, Christopher Konze, and Henner Heck. 2017. Adaptive Cheat Detection in Decentralized Volunteer Computing with Untrusted Nodes. In *Proceedings of the* 17^{th} *IFIP International Conference on Distributed Applications and Interoperable Systems* (Neuchâtel, Switzerland) *(DAIS '17)*. Springer, 192–205. https://doi.org/10.1007/978-3-319-59665-5_14

[34] Martin Kristien, Tom Spink, Harry Wagstaff, Björn Franke, Igor Böhm, and Nigel Topham. 2019. Mitigating JIT Compilation Latency in Virtual Execution Environments. In *Proceedings of the* 15^{th} *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Providence, RI, USA) *(VEE '19)*. Association for Computing Machinery, New York, NY, USA, 101–107. https://doi.org/10.1145/3313808.3313818

[35] Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. 2019. Asynchronous I/O Stack: A Low-latency Kernel I/O Stack for Ultra-Low Latency SSDs. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC '19)*. USENIX Association, Renton, WA, 603–616. https://www.usenix.org/conference/atc19/presentation/lee-gyusun

[36] Pejman Lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer, and Babak Falsafi. 2012. Scale-out Processors. In *Proceedings of the* 39^{th} *Annual International Symposium on Computer Architecture* (Portland, Oregon) *(ISCA '12)*. IEEE Computer Society, USA, 500–511.

[37] Patrick Merias. 2018. *Study on physical layer enhancements for NR ultra-reliable and low latency case (URLLC)*. Technical Report TR 38.824, release 16. 3^{rd} Generation Partnership Project (3GPP).

[38] Nathaniel Morris, Christopher Stewart, Lydia Chen, Robert Birke, and Jaimie Kelley. 2018. Model-Driven Computational Sprinting. In *Proceedings of the* 13^{th} *European Conference on Computer Systems* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 38, 13 pages. https://doi.org/10.1145/3190508.3190543

[39] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. 2020. A First Look at Commercial 5G Performance on Smartphones. In *Proceedings of The Web Conference* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 894–905. https://doi.org/10.1145/3366423.3380169

[40] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. 2018. Understanding PCIe Performance for End Host Networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 327–341. https://doi.org/10.1145/3230543.3230560

[41] Khang T. Nguyen. 2013. APIC Virtualization Performance Testing and Iozone. https://software.intel.com/content/www/us/en/develop/blogs/apic-virtualization-performance-testing-and-iozone.html. Accessed 2020-12-20.

[42] Edmund B. Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt. 2009. Helios: Heterogeneous Multiprocessing with Satellite Kernels. In *Proceedings of the* 22^{nd} *ACM SIGOPS Symposium on Operating Systems Principles* (Big Sky, Montana, USA) *(SOSP '09)*. Association for Computing Machinery, New York, NY, USA, 221–234. https://doi.org/10.1145/1629575.1629597

[43] Stanko Novakovic, Alexandros Daglis, Edouard Bugnion, Babak Falsafi, and Boris Grot. 2014. Scale-out NUMA. In *Proceedings of the* 19^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems* (Salt Lake City, Utah, USA) *(ASPLOS '14)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/2541940.2541965

[44] Open Connectivity Foundation. 2021. UPnP Standard. https://openconnectivity.org/developer/specifications/upnp-resources/upnp/. Accessed 2020-11-05.

[45] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. 2010. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. *SIGOPS Operating Systems Review* 43, 4 (Jan. 2010), 92–105. https://doi.org/10.1145/1713254.1713276

[46] Jiannan Ouyang, Brian Kocoloski, John R. Lange, and Kevin Pedretti. 2015. Achieving Performance Isolation with Lightweight Co-Kernels. In *Proceedings of the* 24^{th} *International Symposium on High-Performance Parallel and Distributed Computing* (Portland, Oregon, USA) *(HPDC '15)*. Association for Computing Machinery, New York, NY, USA, 149–160. https://doi.org/10.1145/2749246.2749273

[47] Tirthak Patel and Devesh Tiwari. 2020. CLITE: Efficient and QoS-Aware Co-Location of Multiple Latency-Critical Jobs for Warehouse Scale Computers. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture* (San Diego, CA, USA) *(HPCA '20)*. IEEE, 193–206. https://doi.org/10.1109/HPCA47549.2020.00025

[48] Arun Raghavan, Yixin Luo, Anuj Chandawalla, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. 2012. Computational Sprinting. In *Proceedings of the* 18^{th} *IEEE International Symposium on High-Performance Computer Architecture (HPCA '12)*. IEEE Computer Society, USA, 1–12. https://doi.org/10.1109/HPCA.2012.6169031

[49] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *Proceedings of the* 14^{th} *USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. USENIX Association, 315–332. https://www.usenix.org/conference/osdi20/presentation/ruan

[50] Ardalan Amiri Sani and Thomas Anderson. 2019. The Case for I/O-Device-as-a-Service. In *Proceedings of the* 17^{th} *Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS XVII)*. Association for Computing Machinery, New York, NY, USA, 66–72. https://doi.org/10.1145/3317550.3321446

[51] Mahadev Satyanarayanan. 2015. A Brief History of Cloud Offload: A Personal Journey from Odyssey Through Cyber Foraging to Cloudlets. *GetMobile: Mobile Computing and Communications* 18, 4 (Jan. 2015), 19–23. https://doi.org/10.1145/2721914.2721921

[52] Mahadev Satyanarayanan, Wei Gao, and Brandon Lucia. 2019. The Computing Landscape of the 21st Century. In *Proceedings of the* 20^{th} *International Workshop on Mobile Computing Systems and Applications* (Santa Cruz, CA, USA) *(HotMobile '19)*. Association for Computing Machinery, New York, NY, USA, 45–50. https://doi.org/10.1145/3301293.3302357

[53] Mahadev Satyanarayanan, Guenter Klas, Marco Silva, and Simone Mangiante. 2019. The Seminal Role of Edge-Native Applications. In *Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE '19)*. 33–40. https://doi.org/10.1109/EDGE.2019.00022

[54] Malte Schwarzkopf, Matthew P. Grosvenor, and Steven Hand. 2013. New Wine in Old Skins: The Case for Distributed Operating Systems in the Data Center. In *Proceedings of the* 4^{th} *Asia-Pacific Workshop on Systems* (Singapore, Singapore) *(APSys '13)*. Association for Computing Machinery, New York, NY, USA, Article 9, 7 pages. https://doi.org/10.1145/2500727.2500739

[55] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *Proceedings of the* 13^{th} *USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*. USENIX Association, Carlsbad, CA, 69–87. https://www.usenix.org/conference/osdi18/presentation/shan

[56] Jan Solanti, Michal Babej, Julius Ikkala, and Pekka Jääskeläinen. 2020. POCL-R: Distributed OpenCL Runtime for Low Latency Remote Offloading. In *Proceedings of the International Workshop on OpenCL* (Munich, Germany) *(IWOCL '20)*. Association for Computing Machinery, New York, NY, USA, Article 19. https://doi.org/10.1145/3388333.3388642

[57] Ya-Yunn Su and Jason Flinn. 2005. Slingshot: Deploying Stateful Services in Wireless Hotspots. In *Proceedings of the* 3^{rd} *International Conference on Mobile Systems, Applications, and Services* (Seattle, Washington) *(MobiSys '05)*. Association for Computing Machinery, New York, NY, USA, 79–92. https://doi.org/10.1145/1067170.1067180

[58] Kaixin Sui, Mengyu Zhou, Dapeng Liu, Minghua Ma, Dan Pei, Youjian Zhao, Zimu Li, and Thomas Moscibroda. 2016. Characterizing and Improving WiFi Latency in Large-Scale Operational Networks. In *Proceedings of the* 14^{th} *Annual International Conference on Mobile Systems, Applications, and Services* (Singapore, Singapore) *(MobiSys '16)*. Association for Computing Machinery, New York, NY, USA, 347–360. https://doi.org/10.1145/2906388.2906393

[59] Andrew S. Tanenbaum and Robbert Van Renesse. 1985. Distributed Operating Systems. *Comput. Surveys* 17, 4 (Dec. 1985), 419–470. https://doi.org/10.1145/6041.6074

[60] Liang Tong, Yong Li, and Wei Gao. 2016. A hierarchical edge cloud architecture for mobile computing. In *Proceedings of the* 35^{th} *Annual IEEE International Conference on Computer Communications* (San Francisco, CA) *(INFOCOM '16)*. IEEE, 1–9. https://doi.org/10.1109/INFOCOM.2016.7524340

[61] Animesh Trivedi, Lin Wang, Henri Bal, and Alexandru Iosup. 2020. Sharing and Caring of Data at the Edge. In *Proceedings of the* 3^{rd} *USENIX Workshop on Hot Topics in Edge Computing (HotEdge '20)*. USENIX Association. https://www.usenix.org/conference/hotedge20/presentation/trivedi

[62] Shin-Yeh Tsai and Yiying Zhang. 2017. LITE Kernel RDMA Support for Datacenter Applications. In *Proceedings of the* 26^{th} *Symposium on Operating Systems Principles (SOSP '17)*. 306–324. https://doi.org/10.1145/3132747.3132762

[63] Nikos Vasilakis, Ben Karel, and Jonathan M. Smith. 2015. From Lone Dwarfs to Giant Superclusters: Rethinking Operating System Abstractions for the Cloud. In *Proceedings of the* 15^{th} *Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association, Kartause Ittingen, Switzerland. https://www.usenix.org/conference/hotos15/workshop-program/presentation/vasilakis

[64] Nicholas Wanninger, Joshua J. Bowden, and Kyle C. Hale. 2021. Virtines: Virtualization at Function Call Granularity. arXiv:2104.11324 [cs.PL]

[65] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. 2021. Q-VR: System-Level Design for Future Mobile Collaborative Virtual Reality. In *Proceedings of the* 26^{th} *ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 587–599. https://doi.org/10.1145/3445814.3446715

[66] Hangchen Yu, Arthur Michener Peters, Amogh Akshintala, and Christopher J. Rossbach. 2020. AvA: Accelerated Virtualization of Accelerators. In *Proceedings of the* 25^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 807–825. https://doi.org/10.1145/3373376.3378466

[67] Gerd Zellweger, Simon Gerber, Kornilios Kourtis, and Timothy Roscoe. 2014. Decoupling Cores, Kernels, and Operating Systems. In *Proceedings of the* 11^{th} *USENIX Symposium on Operating Systems Design and Implementation* (Broomfield, CO, USA) *(OSDI '14)*. USENIX Association, USA, 17–31.

[68] Steffen Zeuch, Eleni Tzirita Zacharatou, Shuhao Zhang, Xenofon Chatziliadis, Ankit Chaudhary, Bonaventura Del Monte, Dimitrios Giouroukis, Philipp M Grulich, Ariane Ziehn, and Volker Mark. 2020. NebulaStream: Complex analytics beyond the cloud. In *Proceedings of the International Workshop on Very Large Internet of Things* (Tokyo, Japan) *(VLIoT '20)*.

[69] Jin Zhang, Zhuocheng Ding, Yubin Chen, Xingguo Jia, Boshi Yu, Zhengwei Qi, and Haibing Guan. 2020. GiantVM: A Type-II Hypervisor Implementing Many-to-One Virtualization. In *Proceedings of the* 16^{th} *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Lausanne, Switzerland) *(VEE '20)*. Association for Computing Machinery, New York, NY, USA, 30–44. https://doi.org/10.1145/3381052.3381324

[70] Wenxiao Zhang, Bo Han, and Pan Hui. 2017. On the Networking Challenges of Mobile Augmented Reality. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network* (Los Angeles, CA, USA) *(VR/AR Network '17)*. Association for Computing Machinery, New York, NY, USA, 24–29. https://doi.org/10.1145/3097895.3097900

[71] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. 2015. Mojim: A Reliable and Highly-Available Non-Volatile Memory System. In *Proceedings of the* 20^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems* (Istanbul, Turkey) *(ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 3–18. https://doi.org/10.1145/2694344.2694370