A Heterogeneous Microkernel OS for Rack-Scale Systems

Matthias Hille
Technische Universität Dresden
matthias.hille@tu-dresden.de

Nils Asmussen
Barkhausen Institut
nils.asmussen@barkhauseninstitut.org

Hermann Härtig
Technische Universität Dresden
hermann.haertig@tu-dresden.de

Pramod Bhatotia
The University of Edinburgh
pramod.bhatotia@ed.ac.uk

ABSTRACT

Datacenters are adopting heterogeneous hardware in the form of different CPU ISAs and accelerators. Advances in low-latency and high-bandwidth interconnects enable hardware vendors to tighten the coupling of multiple CPU servers and accelerators. The closer connection of components facilitates bigger machines, which pose a new challenge to operating systems. We advocate to build a heterogeneous OS for large heterogeneous systems by combining multiple OS design principles to leverage the benefits of each design. Because a security-oriented design, enabled by simplicity and clear encapsulation, is vital in datacenters, we choose to survey various design principles found in microkernel-based systems. We explain that heterogeneous hardware employs different mechanisms to enforce access rights, for example for memory accesses or communication channels. We outline a way to combine enforcement mechanisms of CPUs and accelerators in one system. A consequence of this is a heterogeneous access rights management which is implemented as a heterogeneous capability system in a microkernel-based OS.

ACM Reference Format:
Matthias Hille, Nils Asmussen, Hermann Härtig, and Pramod Bhatotia. 2020. A Heterogeneous Microkernel OS for Rack-Scale Systems. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '20), August 24–25, 2020, Tsukuba, Japan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3409963.3410487

APSys '20, August 24–25, 2020, Tsukuba, Japan

ACM ISBN 978-1-4503-8069-0/20/08…$15.00
https://doi.org/10.1145/3409963.3410487

1 INTRODUCTION

Heterogeneous hardware is finding its way into datacenters. Cloud vendors offer x86 and ARM general purpose cores, GPUs, TPUs and FPGAs [20, 25, 26, 28]. Today these components are hosted in discrete servers which are bundled in a rack [16]. A server consists of multiple CPUs, memory and extension cards including accelerator cards [38]. This design offers little flexibility regarding the ratio between general-purpose and specialized compute power, meaning CPUs and accelerators, respectively. One could argue that there is flexibility within one server, namely whether or not accelerator cards are added and if so, which kind of accelerators. However, this is not the granularity a datacenter operator thinks of. Companies which run large datacenters, like Facebook, are starting to build racks out of chassis which consist of different server types like compute blades and accelerators [4]. This shows, that in a datacenter the flexibility should rather be at the level of a whole rack to fit the demands of individual workloads and ease component composition. From this demand of flexibility we derive an architecture that brings flexibility into a rack by means of different board types which make up a rack.

In a rack-scale design as depicted in Figure 1, a machine handled by the operating system comprises the whole rack. Due to the amount of components and the size of the rack the overhead for cache-coherence across the whole rack is prohibitive. Hence, we do not expect a rack to resemble a cache-coherent machine but rather a conglomeration of cache-coherent islands [37]. This leaves the OS with a heterogeneous system containing various architectures and communication schemes. On top of that, the enforcement of access rights, one of the main tasks of an OS, works differently on diverse architectures.

Various operating systems have been developed and optimized for a particular system type. One such system type are cache coherent systems for which highly optimized operating systems like Linux, Windows, XNU and Fiasco.OC [34] have been developed. on the contrary, there are systems

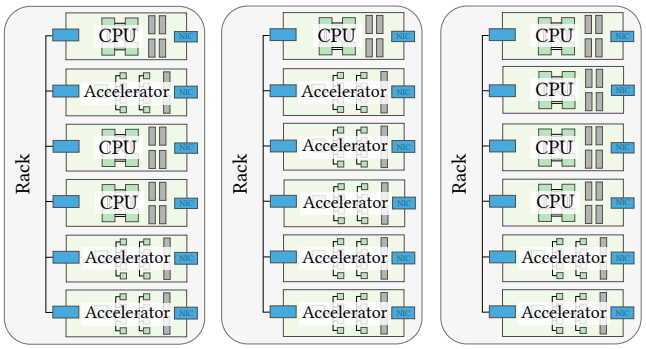

Figure 1: Architectural scheme for racks in a datacenter: A rack is equipped with two different board types: CPU boards and accelerator boards. Each board is connected with an intra-rack interconnect resembled by the blue boxes on the left. The unconstrained choice between board types provides flexibility in this design.

like Popcorn Linux [6] and M^3 [5] designed to work without cache-coherence. The design of the Barrelfish multikernel [7] aims at a shared-nothing architecture, while its implementation still uses cache-coherence protocols to implement messaging. So far the requirements for different system types were following from the integration of CPUs and memory, but there are also accelerators which have been considered in OS designs like Omnix [45] and M^3. They strive to elevate accelerators to first-class citizens in the OS design, giving them direct access to OS services as other systems also tried for selected types of accelerators [46, 47]. Finally, an OS design like LegoOS [43], considering disaggregated hardware resources, splits the OS into various components for each resource type.

In our opinion each of these OS designs solves a specific requirement, but in a flexible rack-scale datacenter design as we envision it, multiple requirements exist in parallel like architectural heterogeneity and the heterogeneity of communication mechanisms. Hence, we propose to pick features of the individual OS designs and combine them into one heterogeneous rack-scale OS.

Besides the goals to improve flexibility, system performance and integrate accelerators into the system, security is essential for datacenters. A cloud datacenter is constantly exposed via the network and security breaches due to erroneous software raise serious threats to a cloud vendor's business. Microkernel-based OS designs suit the increased security requirements due to their small trusted computing base (TCB), clear encapsulation and least-privilege access policy. The small code base of the microkernel makes it possible to verify the kernel [33]. Encapsulation and a least-privilege access policy enable failure containment limiting the severity of many errors compared to the same errors in a monolithic OS [9]. The least-privilege access policy is typically implemented by means of capabilities [34].

We propose to merge different microkernel-based OS designs into one OS to get the maximum out of both: CPUs and accelerators. When combining different OS designs diverse OS abstractions and therefore various capability types and hardware features are fused in one system. This raises the question how to design hardware that couples different hardware architectures in a system so it is modular, secure, and provides isolation which is configurable by software. The combination of different isolation mechanisms requires the OS to control these and mediate resources of different architectures.

In this work we present microkernel-based OS architecture for a new hardware paradigm in large datacenter. We cover the hardware landscape and the resulting aspects for the OS with an emphasis on microkernels. We give an outline how to combine different isolation and communication mechanisms in an OS and the implications on the capability system.

2 DATACENTER ARCHITECTURE

A modular and extensible hardware architecture is crucial for datacenters to facilitate fast exchange of components or quick launches of new hardware. In the following we present a hardware architecture of a rack derived from recent efforts of hardware vendors like Dell and datacenter providers including Facebook, Microsoft and Amazon [16, 23, 36, 48]. We further explain our view on how a suitable OS for such a system can be built.

2.1 Hardware Architecture

We base our architecture on the trend that resources are becoming more numerous and accelerators gain importance [3, 13, 24, 28, 55] and also interconnects advance and enable tighter coupling of servers [23, 48]. Figure 2 depicts a snippet of a rack. We choose a machine to span a whole rack, however, the principles of our approach are also applicable to machines of smaller extent, for example a chassis comprising several boards.

All boards of a rack are interconnected with a fast fabric which enables close collaboration between them. The interconnect provides two main features: 1) configurable communication channels for *message passing* and 2) a configurable access to *memory ranges*. This can be implemented by an interconnect like Gen-Z [15], CXL [1] as has been showcased by Dell EMC recently [32]. Similar to these interconnects M^3 [5] introduced a component called *data transfer unit* (DTU) providing a configurable high-bandwidth interconnect. The interconnect between the boards is configured by a central privileged entity labeled with rack kernel in

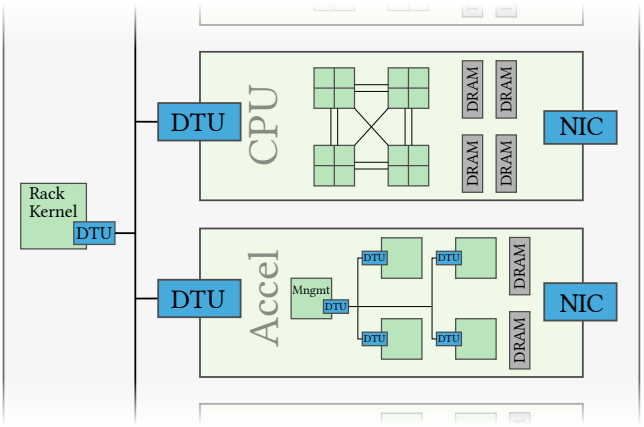

Figure 2: Excerpt of the rack architecture with two board types: CPU boads and accelerator boards. CPUs have cache-coherent access to their board-local DRAM. Accelerators are connected via DTUs. The interconnect hooking up CPUs and accelerators to their board DTU (on the left) are left out for clarity.

Figure 2. A DTU has configurable endpoints for this purpose which can be programmed to be send, receive or memory endpoints. Gen-Z offers mechanisms like Access Keys to ensure isolation between components and Write MSG packets for messaging [15]. At the rack level the OS employs explicit messaging to manage the boards.

The part of the OS running on the boards will send requests to the rack kernel to set up communication channels and access to memory ranges between boards. This might seem overprotective if the whole machine is within one trust sphere, e.g. if one customer rents the whole rack. However, it enables to implement the least-privilege access policy across the whole rack. The overhead for this should be minimal since the communication channels and memory ranges are usable without interaction of the rack kernel, because hardware enforces the access rights once they are set up. Furthermore, this design enables partitioning of the rack so parts of the rack can be used by mutually distrusting parties. Similar to this principle but in the scope of one machine (i.e. one board), Amazon has introduced the hardware supported hypervisor AWS Nitro which is supported by a security chip dedicated to manage virtualization tasks [36]. Furthermore, Zhang et al. [53] have shown that a hardware component, which controls the data flow and communicatiorn, can be used to guaranty information containment.

A rack in our system is made of two basic types of boards: *CPU boards* featuring powerful cache-coherent general purpose cores and *accelerator boards* comprising either GPUs, TPUs, FPGAs or other kinds of accelerators. Both board types are equipped with memory which is also accessible from other boards (via the board DTU). The system could also be extended with a memory or storage board type to integrate NVM and disks into the system. However, in this work we concentrate on compute resources. CPU boards are very similar to today's server systems except that they are connected to other boards via the intra-rack interconnect and they do not incorporate accelerators as extension cards. Our architecture is similar to the Zion accelerator platform recently introduced by Facebook [31]. In the Zion platform dual-socket compute blades and accelerator modules coexist. A proprietary PCIe-based interconnect is used between the compute boards and the accelerator modules. While this interconnect is designed to provide low latency and high bandwidth it is missing the configurability an interconnect with a configurable component like the DTU provides.

Additional to the interconnect within a rack, each board is equipped with a network interface card. Accelerator boards internally employ the same type of interconnect as is used between the boards of a rack. An accelerator board possesses a management CPU running the OS which configures the interconnect for the accelerators. To enable direct access to accelerators or stream data to the network directly accelerator boards also possess a network interface which is driven by the management CPU.

2.2 OS Cherry Picking

Each OS design is targeting different hardware architectures or design goals like security by simplicity, a unified hardware abstraction or legacy compatibility [5, 14, 22]. We want to cherry pick suitable hardware and software features and utilize them to manage the parts of the system for which their design fits best.

Size Does Matter. Today operating systems deployed in datacenters are monolithic. They save mode- and context-switches by putting many OS services into the kernel in contrast to microkernels [35]. However, such a large TCB is hard to maintain and can lead to security issues. Therefore we believe that microkernels are a better fit for datacenters which are constantly exposed via the network and should be secure at all times to protect customer data and business reputation. The small TCB and the clear encapsulation of a microkernel-based OS make it easier to review the system's code base and it becomes affordable to verify important parts of the system like the microkernel itself [33]. Microkernels employ capabilities to implement the least-privilege principle which helps to prevent privilege escalation due to errors in OS services (failure containment). The argument that most microkernel-based OSes are missing the compatibility (e.g. drivers) does not hold in a cloud datacenter because it is a

closed hardware ecosystem with a limited variety of hardware components for which drivers can be maintained with reasonable effort.

Specialization. Among microkernel-based systems there is none which has all advantages one would wish for in a datacenter with large racks consisting of heterogeneous cores and accelerators. There are L4 microkernels like Fiasco.OC, seL4 [33, 34] which are well-suited for powerful cache-coherent general purpose CPUs. A multikernel like Barrelfish [7] uses explicit message passing which makes the design a good fit to manage different cache-coherent islands [8] inside a rack, but it is a radical choice to completely abandon the cache-coherence mechanisms which are there anyways.

Hardware Support. Current COTS systems employ *memory management units (MMU)* and IOMMUs to support hardware-enforced memory isolation. While this design is battle-tested, it requires CPUs to implement privileged mode execution and IOMMUs to be configured by the host CPU. Systems using ISA capabilities like CHERI [52] offer fast delegation of access rights which is a good fit for large systems, however, this comes at the price that revocation is either not possible or is limited to revoking all capabilities to a resource (in CHERI this is memory) as suggested by Achermann et al. [2]. Both hardware mechanisms, MMUs and ISA capabilities, only support memory isolation but not hardware supported message passing. The M^3 system isolates CPUs and accelerators with its DTU which provides both, memory isolation via memory ranges and message passing via configurable communication channels between components. These features enable the execution of untrusted code on accelerators, fast IPC between any accelerator and/or CPU and the provision of OS services to accelerators [5].

3 OS DESIGN CHALLENGES

The ever growing number of compute resources (CPU cores and accelerators) is a phenomenon OS developer have been exposed to for more than a decade [11, 12]. Maintaining cache coherence for large amounts of CPU cores involves considerable overheads and complicates the system's scalability which is a challenge for hardware development [30]. However, we do think that cache coherence will stay, but in a limited scope [37], for example in groups of CPU cores which we call cache-coherent *islands*. The OS which used to leverage a homogeneous communication mechanism to manage the system will then face cache-coherent islands which can use shared-memory communication within islands but explicit messaging to communicate between them.

Resource disaggregation is moving in the focus of the research community these days [29, 39, 43] to integrate numerous resources into a system including the growing number of accelerators [10, 20, 28, 42, 49, 54]. One of the challenges with respect to accelerators is how to integrate them closely into the system to keep the data transfer costs low, which are projected to increase their share regarding energy consumption and execution time [41]. Today large amounts of accelerators are connected with high-speed interconnects [19] and form pods of accelerators which work together closely [21, 27]. However, this limited spatial integration is not suitable for all workloads. Considering this we envision a machine to span a whole rack consisting of boards of powerful general-purpose CPUs, GPUs, TPUs, FPGAs and other accelerators.

In the heterogeneous hardware environment of a rack consisting of different boards, the OS has to handle two kinds of heterogeneity: different ISAs and different communication schemes (cache-coherence vs. explicit messaging).

ISA Heterogeneity. Managing a system consisting of different ISAs requires to execute different low-level implementations of an OS kernel in the same system. These kernel implementations are sharing the same OS design choices, for example they have a unified process management and access-rights management. There are different ways to abstract away the heterogeneous ISAs: one way is to use software only, like Popocorn Linux or Barrelfish do [6, 7] and another is a hardware/software co-design in which a hardware component offers a unified interface to different ISAs and the abstraction is realized in hardware as M^3 does [5]. Some research OSes cannot just handle general purpose cores with different ISAs but also accelerators, which we consider as an extreme but very important case of ISA heterogeneity. The challenge is to build an OS which is versatile enough to handle different ISAs and accelerators without loosing the performance benefits of years of optimization to specific architectures.

Communication Heterogeneity. In order to scale, OSes are optimizing for one communication strategy which is either cache coherence or explicit messaging. In cache-coherent systems OS developers use fine-grained locking and careful alignment of data structures to cache lines [50]. This prevents cache thrashing due to false sharing. Other OS designs employ explicit messaging [5, 7, 51] to communicate between the multiple kernels and OS services running the machine. OS developers will be challenged to mix both communication techniques [43] and optimize algorithms appropriately.

Access Rights Management. In microkernels access rights are managed by means of capabilities. A capability is a combination of: 1) a reference to a resource and 2) the access rights to this resource [17]. Microkernel-based systems are

appealing due to their small trusted computing base and comprehensible security architecture. The small size and clear encapsulation enables the verification of parts of the system and can prevent critical errors already by design [9].

The ways to implement capabilities for access rights management are manifold. A capability system in general enables access rights management and enforcement. The required functionality of a capability system can be categorized as follows:

1) Delegation: The delegation of access rights can be overseen by a privileged entity like the OS *kernel* or a *capability co-processor* which verifies whether it is allowed to delegate a particular capability. Only sparse capabilities are delegated without supervision.
2) Enforcement: A capability system needs to ensure that programs obey to their access rights. First this depends on the kind of resource a capability refers to. For memory the most common enforcement mechanism is an *MMU* which is programmed by the kernel. MMUs are typically combined with partitioned capabilities as used in L4 systems. Alternatively there exist ISA capabilities which are enforced by a *capability co-processor* which checks whether the program has the necessary capabilities loaded into its capability registers. Furthermore, the M^3 system uses a hardware component, the so-called *data transfer unit (DTU)*, which can be configured to give an application access to a range of memory [5].

Another important resource is the inter-process communication (IPC). Typically this is enforced by the *kernel* which mediates between two communicating processes. A special case is the M^3 system [5] containing a DTU which can be configured to establish a communication channel between two applications. In such a system the kernel sets up the channel but is not required to enforce it, hence it is not involved in the actual communication.

Other resources like process management and scheduling are enforced by the kernel since it is the privileged entity in control of these features.
3) Revocation: Access rights also need to be withdrawn which is done by revoking a capability. Revocation can be done in various ways and with differing granularity. The most coarse-grained is to revoke all access rights to a resource in a system, e.g. by changing a version number in the referenced resource. A more flexible way is to introduce indirection objects when capabilities are delegated (e.g. across trust spheres). Invalidating an indirection object then only revokes capabilities pointing to this indirection object but not all capabilities to the resource if multiple indirection objects exist [44]. The most flexible approach is the selective revocation of a capability (and all capabilities delegated from this one). Therefore the system has to track delegation or needs to be able to reconstruct the delegation to find all capabilities which need to be revoked. Note that revocation also requires to reset enforcement mechanisms like TLBs, which cache access rights.

In a heterogeneous system with a mix of enforcement mechanisms, the OS has to manage heterogeneous types of capabilities for the same resource type (e.g. kernel-enforced vs. DTU-enforced IPC). Furthermore, capabilities can cross boards which may require the combination of cache-coherent and message-based coordination in the capability system. Designing an OS which mediates access across boards using heterogeneous capabilities and which coordinates between multiple kernels in the system is a challenging task.

4 A HETEROGENEOUS OS DESIGN

4.1 OS Mechanisms

In an architecture, as we describe it, access rights are managed differently on individual boards. The cores of CPU boards could be equipped with MMUs or with capability co-processors, using partitioned capabilities as in L4 systems or ISA capabilities like in CHERI respectively [34, 52]. To illustrate our design we will consider partitioned capabilities in this work.

Since not all accelerators have an MMU we equip accelerators with a DTU that connects them to the board-local network, similar to a network-on-chip. The DTU enables enforcement of memory access rights by configuration of memory endpoints. In addition it provides explicit messaging which enables to enforce IPC without intervention of the kernel. The kernel running on the management CPU only needs to configure the DTUs to set up the communication channels.

4.2 Rack OS

To enable communication between accelerator boards and CPU boards the memory and IPC enforcement mechanisms need to be compatible across the whole rack. Therefore we merge OS mechanisms of different OS designs into a *Rack OS*. In our Rack OS we employ an L4-like kernel to control CPU boards and an M^3-like kernel to control accelerator boards. On top of that we deploy a rack kernel which sets up sharing and communication between individual boards (cf. Figure 2). In the following we discuss how the cross-board coordination is achieved.

Memory. To share memory between accelerator and CPU boards, memory pages have to be translated to memory ranges which can be used to configure DTUs of accelerators such that they can access memory directly. The L4 kernel operating the CPU boards needs to have a notion of how a DTU works such that it can request the rack kernel to configure the board's DTU appropriately. In the other direction the M^3

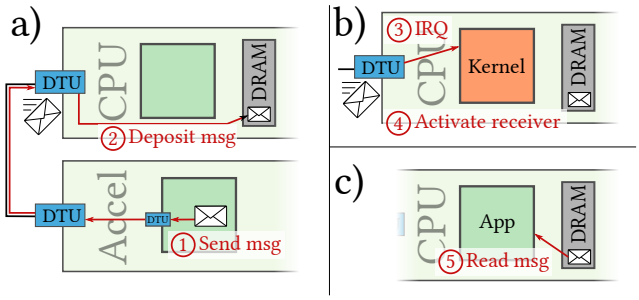

Figure 3: Cross-board communication from an accelerator to a CPU. The accelerator's message is forwarded via the DTUs on the communication channel. The DTU of the CPU board deposits the message in memory and informs the Kernel via an interrupt. The kernel then wakes up the receiver which can now read the message.

kernel operating the accelerator boards has to translate possibly byte-granular memory ranges to memory pages with a fixed page size. To always offer migration between both mechanisms, memory ranges would need to be page-sized as well, which would abandon the flexibility advantage of memory ranges. A solution to this could be to request a special type of memory (migratable memory) which is always page-sized and use byte-granular memory ranges for non-migratable memory. From the OS' perspective this principle extends the notion of pinned and non-pinned memory in today's systems.

Communication. The mediation of IPC between different board types involves a transition from hardware-enforced IPC to kernel-enforced IPC. Figure 3 illustrates the steps of this transition. When an accelerator wants to send a message to a CPU, the accelerator initiates the send operation via its DTU depicted in step ① in Figure 3.a). The message is forwarded by the accelerator's DTU to the DTU of the corresponding accelerator board which forwards it to the CPU board. In step ② the DTU of the CPU board deposits the message in a predestined memory location. Subsequently the DTU signals the arrival of a message to the kernel of the CPU via an interrupt in step ③ in Figure 3.b). The kernel then identifies the receiver and activates the process in step ④. Finally step ⑤ in Figure 3 depicts the receiving process reading the message from memory. For short messages the message can be injected directly into the CPU's registers to minimize latency caused by memory reads and writes, as is common practice in microkernels [18]. For IPC from a CPU board to an accelerator the L4 kernel has to forward the message from the sender running on a CPU via the CPU board's DTU to an accelerator board. Both scenarios require that the boards are actually allowed to communicate with each other and that the boards' DTUs have been set up by the rack kernel. The coordination via messages between the boards' kernels exhibits that we also employ OS design principles known from the multikernel approach [7].

4.3 Heterogeneous Capability System

Each breed of OS kernel used in the Rack OS has its own capability types. This is because they employ different enforcement mechanisms for memory access rights and IPC. In addition to that, execution contexts are disparate due to the diverse compute hardware. A CPU has a vastly different execution context from a GPU or an FPGA. Thus some capability types like a scheduling capability for a CPU thread are not applicable on an accelerator board. However, it can still be useful to have the ability to activate a CPU thread, for example when an accelerator wants to signal an event to a CPU thread.

The resource heterogeneity is reflected in a heterogeneous capability system. There are capabilities which have to be mapped between different implementations (e.g. memory, IPC) and capabilities which only reference remote resources (e.g. processes / execution contexts, scheduling contexts). To enable the delegation of access rights to resources across the whole system, kernels of various boards need to cooperate and exchange capabilities via messages. Hence, our system comprises a distributed capability system similar to the one used in the multikernel approach. However, inside of a CPU board cache coherence can be used to coordinate capability operations between kernel threads running on different cores. The capability system is therefore not only heterogeneous regarding the capability types but also with respect to the coordination of capability operations.

4.4 Additional Challenges

The OS design we propose is a combination of various operating systems into one system. It entails different kernels running on the boards comprising a rack. To manage the system as a whole we gave an outline how to combine different isolation and enforcement mechanisms and the corresponding capability system. However, there are more building blocks of the system which we want to discuss briefly.

Memory Allocation. The memory allocation in a system with memory distributed on the various boards requires the coordination of local per-board allocators. The choice where to allocate memory and when to migrate it to another board can be left to the application developer or a runtime system. In any case the OS has to take care that memory access rights are reflected correctly with both enforcement mechanisms. A migration might be possible using the migratable-memory approach described in section 4.2. An alternative approach

is the integration of ISA capabilities which can be used to enforce byte-granular memory access rights in page-based systems [52]. However, to preserve the capability system's full functionality a mechanism to selectively revoke ISA capabilities would be necessary.

Scheduling. Scheduling in such a large system requires a mixture of board-local schedulers and a mechanism to trigger scheduling of an execution context on another board, to optimize for latency-critical operations for instance. Board-spanning scheduling requires to introduce notions of the various execution contexts into every kernel of the system. This is also useful for setting up accelerator tasks from a CPU thread or vice versa to dynamically generate CPU tasks during computation on an accelerator. The notion of an execution context also affects migration between differing ISAs or even accelerators. Previous work suggests to provide multi-ISA binaries [40] which could be extended to multi-accelerator binaries containing diverse implementations for the same problem.

Resource Management. Resource discovery and representation is another topic where various trade-offs can be found. In a rack consisting of many boards it might be unnecessary that each kernel on every board knows all the resources of the rack. Instead resource discovery can be done by the rack kernel and cross-board resource requests are be mediated by this central entity. Discovery of OS services can work in a similar fashion, where some services might not be usable across boards but others are. For example an in-memory filesystem service, which is designed to utilize the byte-granular memory ranges of a DTU, is not usable in the same way on a CPU board with MMU enforced page-granular memory protection.

5 CONCLUSION

To improve the scalable integration of general purpose cores and accelerators in datacenters we propose to combine multiple OS designs into a Rack OS to leverage the benefits of each design. Extending the isolation mechanisms of accelerators facilitates the cooperation with powerful general purpose boards and enables the OS to provide OS services to accelerators directly. We suggest to base a datacenter OS on microkernels offering a security-oriented design with clear encapsulation and a least-privilege access policy. The combination of multiple OS designs entails merging of different isolation and enforcement mechanisms resulting in a heterogeneous capability system. The operation of such a capability system requires to mediate between varying enforcement mechanisms and the combination of two coordination methods—cache-coherence and explicit messaging.

REFERENCES

[1] Compute Express Link (CXL) Promoters 2019. Compute Express Link Specification 1.0, 2019.

[2] Reto Achermann, Robert N. M. Watson, Chris Dalton, Paolo Faraboschi, Moritz Hoffmann, Dejan Milojicic, Geoffrey Ndu, Alexander Richardson, Timothy Roscoe, and Adrian L. Shaw. Separating translation from protection in address spaces with dynamic remapping. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*, 2017.

[3] Sandeep R Agrawal, Sam Idicula, Arun Raghavan, Evangelos Vlachos, Venkatraman Govindaraju, Venkatanathan Varadarajan, Cagri Balkesen, Georgios Giannikis, Charlie Roth, Nipun Agarwal, and Eric Sedlar. A many-core architecture for in-memory data processing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[4] Alexey Andreyev. The New Facebook DC Topology, 2019. OCP Summit 2019.

[5] Nils Asmussen, Marcus Völp, Benedikt Nöthen, Hermann Härtig, and Gerhard Fettweis. M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[6] Antonio Barbalace, Binoy Ravindran, and David Katz. Popcorn: a replicated-kernel OS based on Linux. *Ottawa Linux Symposium (OLS)*, 2014.

[7] Andrew Baumann, Paul Barham, Pierre-evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP)*, 2009.

[8] Andrew Baumann, Chris Hawblitzel, Kornilios Kourtis, Tim Harris, and Timothy Roscoe. Cosh: clear OS data sharing in an incoherent world. In *2014 Conference on Timely Results in Operating Systems (TRIOS)*, 2014.

[9] Simon Biggs, Damon Lee, and Gernot Heiser. The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security. In *9th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*, 2018.

[10] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference (SIGCOMM)*, 2013.

[11] Silas Boyd-wickizer, Austin T Clements, Yandong Mao, Aleksey Pesterev, M Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. An Analysis of Linux Scalability to Many Cores. *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI)*, 2010.

[12] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for GPUs: Stream computing on graphics hardware. In *ACM SIGGRAPH*, 2004.

[13] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. A cloud-scale acceleration architecture. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[14] Xiaoxin Chen, Tal Garfinkel, E. Christopher Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan R.K. Ports. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[15] Gen-Z Consortium. Gen-z core specification version 1.0, 2018.

[16] Microsoft Corporation. How microsoft designs its cloud-scale servers. https://www.microsoft.com, 2014.

[17] Jack B. Dennis and Earl C. Van Horn. Programming semantics for multiprogrammed computations. *Communications of the ACM*, 1966.

[18] Kevin Elphinstone and Gernot Heiser. From L3 to seL4 what have we learnt in 20 years of L4 microkernels? In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP)*, 2013.

[19] D. Foley and J. Danskin. Ultra-performance pascal gpu and nvlink interconnect. *IEEE Micro*, 2017.

[20] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steve Reinhardt, Adrian Caulfield, Eric Chung, and Doug Burger. A configurable cloud-scale dnn processor for real-time ai. In *Proceedings of the 45th International Symposium on Computer Architecture (ISCA)*, 2018.

[21] N. A. Gawande, J. B. Landwehr, J. A. Daily, N. R. Tallent, A. Vishnu, and D. J. Kerbyson. Scaling deep learning workloads: NVIDIA DGX-1/Pascal and Intel Knights Landing. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017.

[22] H. Härtig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter. The nizza secure-system architecture. In *2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2005.

[23] Robert Hormuth. Dell EMC's 2019 Server Trends & Observations. https://blog.dellemc.com/en-us/dell-emc-s-2019-server-trends-observations, 2019.

[24] Muhuan Huang, Di Wu, Cody Hao Yu, Zhenman Fang, Matteo Interlandi, Tyson Condie, and Jason Cong. Programming and runtime support to blaze fpga accelerator deployment at datacenter scale. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SoCC)*, 2016.

[25] Amazon Web Services Inc. Amazon elastic graphics. https://aws.amazon.com/ec2/elastic-graphics/.

[26] Google Inc. Cloud TPUs – ML accelerators for TensorFlow. https://cloud.google.com/tpu/.

[27] Google Inc. System architecture cloud tpu. https://cloud.google.com/tpu/docs/system-architecture.

[28] Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[29] Katrinis et al. Rack-scale disaggregated cloud data centers: The dredbox project vision. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016.

[30] Stefanos Kaxiras and Alberto Ros. A new perspective for efficient virtual-cache coherence. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013.

[31] Patrick Kennedy. Facebook zion accelerator platform for oam. https://www.servethehome.com/facebook-zion-accelerator-platform-for-oam.

[32] Patrick Kennedy. Gen-z in dell emc poweredge mx and cxl implications. https://www.servethehome.com/gen-z-in-dell-emc-poweredge-mx-and-cxl-implications.

[33] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles (SOSP)*, 2009.

[34] Adam Lackorzynski and Alexander Warg. Taming subsystems: Capabilities as universal resource access control in L4. In *Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems (IIES)*, 2009.

[35] Jochen Liedtke. On µ-kernel construction. In *Proceedings of the fifteenth ACM symposium on Operating systems principles (OSDI)*, 1995.

[36] Anthony Liguori. C5 instances and the evolution of amazon ec2 virtualization, 2018.

[37] Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin. Why on-chip cache coherence is here to stay. *Communications of the ACM (CACM)*, 2012.

[38] Open Compute Project Microsoft Corporation. Project olympus 1u server mechanical specification. https://www.opencompute.org/wiki/Server/ProjectOlympus, 2017.

[39] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys)*, 2018.

[40] Pierre Olivier, Sang-Hoon Kim, and Binoy Ravindran. Os support for thread migration and distribution in the fully heterogeneous datacenter. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems - HotOS '17*, 2017.

[41] Ardavan Pedram, Stephen Richardson, Mark Horowitz, Sameh Galal, and Shahar Kvatinsky. Dark memory and accelerator-rich system optimization in the dark silicon era. *IEEE Design and Test (IEEE D&T)*, 2017.

[42] B. Poudel, N. Kumar Giri, and A. Munir. Design and comparative evaluation of gpgpu- and fpga-based mpsoc ecu architectures for secure, dependable, and real-time automotive cps. In *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017.

[43] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. LegoOS: a disseminated , distributed OS for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[44] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. Eros: A fast capability system. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles (SOSP)*, 1999.

[45] Mark Silberstein. OmniX: an accelerator-centric OS for omni-programmable systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*, 2017.

[46] Mark Silberstein, Bryan Ford, Idit Keidar, and Emmett Witchel. GPUfs: Integrating a File System with GPUs. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[47] Hayden Kwok-Hay So and Robert Brodersen. A unified hardware/software runtime environment for fpga-based reconfigurable computers using borph. *ACM Transaction of Embedded Computing Systems (TECS)*, 2008.

[48] Siamak Tavallaei, Whitney Zhao, Tiffany Jin, Cheng Chen, and Richard Ding. OCP Accelerator Module (OAM), 2019. OCP Summit 2019.

[49] Mellanox Technologies. Bluefield multicore system on chip. http://www.mellanox.com/related-docs/npu-multicore-processors/PB_Bluefield_SoC.pdf.

[50] Qi Wang, Yuxin Ren, Matt Scaperoth, and Gabriel Parmer. SPeCK: a kernel for scalable predictability. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015.

[51] David Wentzlaff and Anant Agarwal. Factored operating systems (fos): The Case for a Scalable Operating System for Multicores. *ACM SIGOPS Operating Systems Review*, 2009.

[52] Jonathan Woodruff, Robert N M Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. The cheri capability model: Revisiting risc in an age of risk. In *Proceedings of the 41st International*

Symposium on Computer Architecture (ISCA), 2014.

[53] Hansen Zhang, Soumyadeep Ghosh, Jordan Fix, Sotiris Apostolakis, Stephen R Beard, Nayana P. Nagendra, Taewook Oh, and David I August. Architectural Support for Containment-based Security. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[54] Peng Zhang, Jianbin Fang, Canqun Yang, Tao Tang, Chun Huang, and Zheng Wang. MOCL: An efficient openCL implementation for the Matrix-2000 architecture. In *Proceedings of the 15th ACM International Conference on Computing Frontiers (CF)*, 2018.

[55] Whitney Zhao, Siamak Tavallaei, Richard Ding, and Tiffany Jin. OCP Accelerator Module (OAM) System: An Open Accelerator Infrastructure Project, 2019. OCP Summit 2019.