Towards Practical Multikernel OSes with MySyS

Yauhen Klimiankou
Temporary Unaffiliated
y.klimiankou@protonmail.com

ABSTRACT

The multikernel operating system architecture appeared over a decade ago in response to a significant shift in an underlying computer system architecture that saw a massive transition from unicore to multicore processors to deal with scalability and hardware diversity issues. Multikernel considers the computer a distributed system, directly reflects this observation to the OS design and proposes several respective design principles. In particular, multikernel OS constructs the system as a tightly-coupled distributed operating system consisting of a network of isolated nodes and interconnections between them. Furthermore, it replaces implicit inter-core communication through shared memory by explicit message passing and maintains a general OS state using replication. The original multikernel design was prototyped in the experimental operating system Barrelfish and proved that it scales better than conventional Linux in environments of many-core computer systems.

While over a decade has passed after the multikernel OS design introduction, we still failed to see any significant application of its ideas outside of academic research. We reevaluate the architecture of multikernel OS and their design principles from the practical application viewpoint and consider the current state of the art in the microprocessor market, reconsidering assumptions made ten years ago. We highlight the advantages of multikernel OS design and the problems preventing its industrial adoption. Finally, we propose a practical way to overcome these challenges and promote beneficial practical applications of the new OS architecture.

ACM Reference Format:
Yauhen Klimiankou. 2022. Towards Practical Multikernel OSes with MySyS. In *13th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '22), August 23–24, 2022, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3546591.3547525

APSys '22, August 23–24, 2022, Virtual Event, Singapore

ACM ISBN 978-1-4503-9441-3/22/08.
https://doi.org/10.1145/3546591.3547525

1 INTRODUCTION

In the middle of the 2000s computer industry experienced a dramatic shift of a threefold nature. Firstly, AMD initiated a massive transition from 32-bit to 64-bit architectures in Spring 2003 with its Opteron [31] and Athlon 64 processors, while Intel supported it soon in June 2004 with Xeon and Pentium 4 processors. Secondly, VMware revived interest in virtualization by bringing it to the x86 computers in 1999 [55] and servers in 2001 [60]. Soon, in 2005, inspired by the fast and widespread application of virtualization technologies, Intel introduced ISA extension for hardware-assisted virtualization – VMX [58], while AMD replied to Intel in 2006 with SVM [1]. Finally, both major x86 CPU vendors hit the Brick Wall (Power Wall + ILP Wall + Frequency Wall + Memory Wall), and Intel failed with NetBurst and Hyper-Pipelined Technology (deep pipelining) [10]. In response, in 2005, AMD initiated the industry-wide transition to Chip Multiprocessing [46] with their dual-core Athlon 64 x2 and Opteron CPUs [20, 43]. If in the 1990s, computers were equipped with 32-bit single-core processors without virtualization support, then in the 2010s, the vast majority of them were equipped with 64-bit processors supporting virtualization and having from 2 up to 64 cores per chip.

This quick and massive shift in underlying computer architecture has created new challenges and opportunities for software development in general and operating systems designers in particular. The OS research community answered, to a large extent, the challenge of transition to 64-bit architectures back in the 90s based on early non-x86 64-bit processors like MIPS [44], DEC Alpha [53], and UltraSPARC [23] with single address space operating systems (SASOS) like Opal [16], Mungi [26], and Nemesis [25]. However, SASOS never have found any significant application in the industry (some embedded systems are an exception). Furthermore, some features of x86-64 ISA prevent the benefits of SASOS.

In contrast to SAMOS, virtualization technologies raised a hype in the system researcher's community and became a hot topic for years [1, 4, 17, 49, 54, 58, 60]. Furthermore, virtualization technologies became a hot topic not only in academic environment but substantially changed the landscape of computer industry and became the standard infrastructure for data centers [32]. Moreover, virtualization opened the way to a new computing paradigm – cloud computing [13, 14, 61] which use and development still continuously intensify.

The transition from conventional single-core to multi-core x86 processors initially took place without significant attention from the system research community. Conventional and widespread operating systems like Linux and Windows initially took the conservative path of adaptation to the new multi-core environment by reusing already existing facilities for SMP with minimal additional investments [51]. These operating systems developed and implemented these facilities during the 1990s to support high-performance multiprocessor servers and reused them for a smooth transition to multi-core processors. However, soon, system researchers started to be concerned about operating systems scalability [19, 52, 56], and processor design researchers boosted these concerns by projections of soon arrival of heterogeneous [24, 27, 35] and many-core CPUs [11, 29, 42, 50]. This pressure forced them to notice the principal shift in computer architecture – computers from conventional centralized computational systems became distributed systems [7]. In response to this observation, the radically new operating system design was introduced [6].

Multikernel is structuring the operating system as an interconnected network of independent nodes, each of which manages its own logically isolated and independent abstract computer, consisting of a CPU core, a memory area, and a set of I/O devices. It assumes no inter-core sharing and moves OS functionality to a distributed system of processes communicating via message-passing. Multikernel OS architecture is basing on three design principles:

1. Make all inter-core communication explicit by using asynchronous message passing instead of shared memory access as a primary means of communication between CPU cores and operating system kernels.

2. Make OS structure hardware-neutral by separating CPU core management code and inter-core communication management code.

3. View the state as replicated instead of shared. Multikernel assumes a unified OS with state shared through replicated on each node.

Multikernel design became prototyped in Barrelfish OS for x86-64 microprocessors [6]. All inter-core communication in Barrelfish occurs by message passing facilities, which implementation is tailored carefully to the cache coherency protocol. OS supports the traditional process model of threads sharing a single virtual address space across multiple cores (Single System Image or SSI).

Multikernel in general and Barrelfish, in particular, have inherited heavily from early Distributed Operating Systems (DOS) developed in the 1980s for loosely-coupled networks of computers [57] but reworked DOS ideas for a new environment of tightly-coupled multi-core processors (heterogeneous) with shared memory. The primary motivation behind Barrelfish was dealing with scalability and hardware diversity issues of the operating system in the context of heterogeneous multi- and many-core processors, and it proved that with the growth of cores count, it scales better than conventional operating systems.

Barrelfish motivated a lot of OS research in the domain of multikernel design [3, 8, 9, 21, 28, 36, 47, 59]. However, after more than a decade passed, ideas of multikernel design still fail to go beyond the academic world and find any significant application in the industry [9]. In this paper we reevaluate ideas and assumptions behind multikernel and propose a path to bridge the gap between academic research and industrial application of multikernel.

The paper continues as follows: Section 2 provides an overview of the multikernel operating system projects. Section 3 provides a discussion on a reevaluation of multikernel architecture and design principles. We consider the missed benefits of a multikernel in Section 4. Finally, we propose a way for bridging gap between multikernel and real-world applications in Section 5 and conclude in Section 6.

2 MULTIKERNEL OPERATING SYSTEMS

Ideas of multikernel design ideas fell on fertile soil and motivated considerable academic research worldwide. Below we briefly discuss the most notable of them.

Quest-V is a multikernel operating system from Boston University [36]. Like Barrelfish, Quest-V is structured as a distributed system on a chip and linked into a single entity by asynchronous message passing via shared memory channels. In contrast to Barrelfish trying to solve the scalability and hardware diversity problems, Quest-V aimed for a high-confidence multikernel. Because of this, Quest-V targets real-time and mission-critical application domains and exploits ideas of multikernel to provide extra reliability and fault tolerance. For that purpose, Quest-V accompanies multikernel concepts by using hardware virtualization mechanisms for the partitioning of kernel services and by ensuring their isolation. Quest-V does not support the concept of SSI. It prohibits sharing the processes by different CPUs but allows the sharing of device driver data structures to avoid complex I/O virtualization.

Clustered Multikernel from NICTA [59] adopts the ideas of multikernel OS design to deal with the scalability issue of seL4 microkernel [33]. seL4 is a minimalist concurrency-free microkernel designed to keep kernel verification complexity at an acceptable level. For concurrency avoidance, seL4 employs a big kernel lock, which leads to a scalability bottleneck. The more threads run in parallel, the more contention they will experience accessing kernel services. The Clustered Multikernel statically divides the computer system into clusters, each encompassing a set of cores and memory regions controlled by its private instance of the concurrency-free kernel. A system administrator can set the appropriate trade-off between kernel scalability and application-level parallelism by managing the number and size of clusters. At the same time, kernel verification complexity stays almost the same. Like Quest-V, clustered multikernel does not support SSI abstraction.

Popcorn Linux from Virginia Tech [3] is a replicated-kernel OS that aims to apply the multikernel OS design to the traditional operating system, such as Linux. Popcorn Linux is an SSI operating system built on top of multiple instances of a kernel image, such that every kernel maintains its state and manages its resources. Userspace applications can run on it transparently without modification. Popcorn implements software-based coherency based on inter-kernel communication asynchronous message passing and enables OS state consistency. Applications can exploit replication at the user level too. The Popcorn Linux goal is to provide a Linux-based SSI environment for heterogeneous and non-coherent computer systems.

IX from Stanford and EPFL [8] is a control plane/data plane multikernel OS designed for high throughput and low latency networking applications. IX extends the Linux to co-locate specialized dataplane OS nodes, which can run networking applications with increased performance requirements. IX employs hardware-assisted virtualization to isolate dataplane nodes. Control plane Linux thus converts into a hypervisor of a specific kind.

Arrakis from the University of Washington [47] is a control plane/data plane multikernel OS similar to IX. In contrast to IX, Arrakis is not an extended Linux but a modified Barrelfish multikernel and leverages hardware support to remove kernel mediation from the data plane. Thus, in Arrakis, a dataplane is an application running with direct access to network interfaces.

IHK/McKernel is not a classical multikernel but a dual-kernel OS from RIKEN [22] designed for Fugaku supercomputer. IHK/McKernel colocates in the single operating system side-by-side two kernels: the general-purpose IHK Linux and the special-purpose LWK McKernel aiming for application in High-Performance Computing. Its design pursues the following goals: provide scalable and consistent execution of large-scale parallel applications, rapidly adapt to exotic hardware and new programming models, provide efficient memory and device management, eliminate OS noise, and support the full POSIX API. IHK/McKernel uses a multikernel approach to isolate Linux and offload the uncritical part of the application from the LWK to the Linux domain. IHK Linux plays a role of a service provider and a big universal driver for everything. In its turn, McKernel offloads most of the POSIX system calls to the Linux kernel while focusing on specific requirements of HPC. IHK/McKernel supports the SSI abstraction by running the proxy process instance on IHK for each process instance executing on LWK.

Rack OS is a recent proposal from the TU Dresden [28]. It combines several different specialized kernels (L4-like[33] and M^3-like[2]) linked by messaging channels to manage Rack with multiple discrete servers bundled together. Rack OS borrows heavily from multikernel ideas to deal with hardware diversity and the absence of cache coherency between different boards. Rack OS considers a new type of computer architecture, taking a position between classical distributed operating systems and classical multikernel.

NrOS from the University of Utah and VMware Research [9] presents another approach to the multikernel – NRkernel model that replicates the kernel but allows replica sharing among cores, balancing performance and simplicity. NrOS constructs from a simple, sequential OS kernel with no concurrency. It scales across NUMA nodes using node replication and uses operation logs to maintain strong consistency between replicas. NrOS was written from scratch to find a new design point in dealing with the complexity and scalability of operating systems.

3 MULTIKERNEL ASSUMPTIONS

In this section, we reconsider and reevaluate the assumptions underlying the original multikernel design.

Systems are increasingly diverse. Systems are not only increasingly diverse but increasingly complicated. The hype of recent years around Spectre [34] and Meltdown [38] vulnerabilities supports this point. The growth rate of ISA features is worrying [5]. Modern CPUs are very diverse under the hood (for instance, compare Intel Core i9 and ARM Cortex-M0). However, the multikernel OS design provides only limited support in dealing with problems of such kind. At the same time, the problem is not new and was discussed and considered by Liedtke in application to microkernel design in the mid-1990s [37]. Liedtke's reply in the form of non-portable architecture-specific implementation of performance-critical code wrapped by portable architecture-agnostic interfaces seems to stay actual and be more appropriate to the diversity problem.

Cores are increasingly diverse. This assumption is still unfulfilled by the industry to a large extent. There are only a few single-ISA heterogeneous CPUs. ARMs big.LITTLE is one example. Recently released Intel's Alder Lake is another one. However, traditional OS can conservatively support this heterogeneity type by updating the OS scheduler. At the same time, while the specialization of processor cores seems attractive, moving this way is blocked by many challenges. Furthermore, the industry faces a "Chicken or the egg" problem moving this way.

The interconnect matters. It is a well-known fundamental property of distributed systems. However, it seems we forgot a lesson learned from the history of distributed operating systems [57]. They had attempted to present to the user a network of computers as a single machine and failed, while distributed applications succeeded ([18, 30] as examples). The forgotten lesson is next: to achieve high performance, the distributed applications should be explicitly adjusted to the target distributed environment at design time to maximize the locality between computations and data and minimize the amount and distance of the communications. The SSI abstraction hides the details of the distributed nature of the underlying machine from the application designers. Therefore, it is unclear how multikernel will achieve efficiency while hiding host environment details behind SSI abstraction.

Messages cost less than shared memory. Barrelfish has firmly proved this claim [6]. However, recent research argues that synchronization over shared memory outperforms messaging-based synchronization at the intra-NUMA-node level, while at the inter-NUMA-node level, the opposite is true [9].

Cache coherence is not a panacea. During the Barrelfish design time, there were significant doubts about cache coherency ability to scale, and processor designers experimented with non-cache-coherent computer architectures [29]. However, as we see, cache coherency is still with us and shows no signs of imminent departure.

Messages are getting easier. The message-passing and event-driven programming model found extensive use in different domains and even programming languages [15]. Furthermore, both of them are especially common in distributed systems programming. These programming models naturally fit into the networked structure of the underlying hardware, while networks are commonly base on packet switching. However, these programming models contradict sticking to SSI abstraction to a large extent and complicate porting of existing applications written for a shared memory environment.

Traditional operating systems will not scale well on many-core computers. One of the principal aims of the original multikernel OS design was dealing with the OS scalability on upcoming many-core computer systems. There was an assumption that traditional OSes based on a monolithic kernel would fail to scale well with the number of cores growing. But almost immediately after Barrelfish's release, the Linux scalability analysis showed the falsehood of this assumption [12].

Summary. Structuring the OS as a distributed system linked by explicit inter-core communication still looks attractive. However, the assumptions about the massive introduction of heterogeneous and non-cache-coherent processors stay unfulfilled. We also express skepticism about SSI abstraction in distributed environments. While this abstraction can simplify the porting of applications to a new system, it also limits their performance and scalability.

4 MISSED VALUES OF MULTIKERNEL

Many researchers of multikernel were focusing on scalability and hardware heterogeneity issues. Meantime, we speculate that we missed other benefits of multikernel OS design. These benefits came directly from the distributed and decentralized nature of multikernel OS. We guess they can be even more valuable and lead to a practical adoption of multikernel.

Horizontal virtualization and software heterogeneity. Multikernel design resembles a distributed networking system on a single computer by the logical division of principal resources: CPU, RAM, and I/O. It provides an opportunity to build a more flexible and diverse operating system. We can co-locate different scheduling and interrupt handling policies, kernels, and subsystems on the same machine, providing various APIs, each optimized for a particular range of workloads. At the same time, we can consider an OS node in a multikernel as a virtual machine or specialized container and design, build, and deploy specific complex applications co-designed and bundled with dedicated OS facilities. Such horizontal or soft virtualization poses a new point in the existing rapidly developing design space [4, 39, 41, 54]. Application packaged with required OS infrastructure into the multikernel OS node has isolated performance and pool of resources and can bring all necessary application-specific OS policies with itself.

Reliability and Fault tolerance. The traditional OSes have a vertical structure. In this structure, the entire software stack relies on a single monolithic system block – kernel. The kernel is the only and the required mediator between computer hardware and software. Due to this, it is a critical system component, and failure automatically means a collapse of the whole system. Multikernel OS adds a horizontal dimension to the system structure. Multikernel resembles a distributed system and inherits the decentralization property from it. A decentralized multikernel OS does not have a single critical component. Instead, it relies on a kernel layer organized as an interlinked network of OS nodes and distributes this criticality between them all. Kernel crash in multikernel leads to loss of only a part of the system but not to the total system failure. There is a tremendous amount of techniques for reliability and fault tolerance accumulated in the domain of distributed systems in general and, specifically, in distributed OSes [45]. Multikernel OS can inherit most of them and brings a previously impossible level of reliability and fault tolerance into the context of a single computer system. Besides, in contrast to the computer network, the computer represents a case of a tightly integrated system with shared memory. This feature creates an opportunity for very flexible and rapid recovery after node failures. Moreover, it allows for minimizing loss in the application state, which is not possible in the case of computer networks.

Full software stack dynamism The critical nature of the kernel in traditional vertically structured operating systems makes it a static component of the system. OS kernel on-the-fly replacement or updating is challenging. Multikernel OS design allows breaking these statics with far going consequences. First of all, the *temporal dynamism* of the kernel becomes possible. Multikernel systems can gradually and seamlessly evolve over a long time distance by adding new features to the OS kernels, fixing bugs and security vulnerabilities inside them, and without the requirement to stop system functioning. Secondly, multikernel opens an opportunity for flexibility generated by *spatial dynamism*. Multikernel systems can rapidly adapt and optimize themselves to changing workloads. The OS can dynamically reload CPU cores with kernels most appropriate for the currently running applications and the computational workload generated by these applications. Furthermore, the OS gets the opportunity to change itself not only qualitatively but also quantitatively. For example, it can collapse by offloading and shutting down processor cores for power management purposes but rapidly restore full computational power in reply to changes in workload or the surrounding environment.

5 BRIDGING THE GAP TO REAL WORLD

We propose a path to practical adoption of a multikernel design, which consists of four major building blocks:

1. Replace homogeneous OS for heterogeneous hardware with the heterogeneous OS for homogeneous hardware. The network of kernels should include different kinds of OS kernels, some of which should be application-specific and provide benefits for the resolution of actual real-world problems. Heterogeneous OS on homogenous hardware assumes that we can collocate radically different OS kernels in the same multikernel network. Multikernel design brings a new dimension into the operating systems design space, which is not competing but orthogonal to the existing OS designs. OS architecture became two-level: system-wide and intra-node. System-wide architecture defines only coarse-grained inter-kernel resource sharing/isolation conventions and communication protocols. Intra-node architecture, in its turn, specifies how locally running applications share and use resources assigned to the OS node.

2. Extend legacy OS to multikernel. Multikernel OS should employ legacy widely used OS kernels such as Linux or Windows. On the one hand, a multikernel can use a conventional OS kernel as a big universal driver for all hardware. On the other hand, it can employ a legacy OS kernel as a service provider and offload complex uncritical tasks. Furthermore, such an OS kernel can serve as a system monitor and toolbox for the host computer administration. Finally, it can play a role of a multikernel OS bootstrapper.

3. Reject SSI as a fundamental abstraction. Multikernel OS implementations employ SSI as a fundamental abstraction primarily for backward compatibility with existing software. This abstraction hides the distributed nature of the underlying system from overlying software. Hence, already existing third-party software can be easily ported and run in the seems natural environment. This approach is similar to distributed shared virtual memory systems in computer networks. However, "despite much work on distributed shared virtual memory systems, performance, and scalability problems have limited their widespread use in favor of explicit message-passing models" [6]. Meantime, software heterogeneity of multikernel OS through the employment of legacy OS kernel solves the same issues with a lack of software, but more elegantly.

4. Use special-purpose computer systems as a primary target for application of multikernel OS ideas. The widespread adoption of multikernel design in the general-purpose OS requires tremendous investments. The requirement of such one-moment huge investments significantly restains both hardware and software vendors in moving forward and experimenting with new architectures. However, a wide range of current special-purpose computer systems can employ multikernel with relatively small investments and gain valuable benefits. Meantime, the development of multikernel design on special-purpose computers ground in a long-term perspective can provide a smooth transition for the new architectures for both hardware and software vendors, stretching over time and between vendors the required investments.

6 MYSYS FRAMEWORK

We propose MySyS, a new approach for multikernel-style application development and deployment. The MySyS framework consists of a deployment application, a multikernel hypervisor, and application bundles.

MySyS can transform non-modified legacy OSes such as Windows or Linux into special-purpose application-specific multikernel on the fly, dynamically deploy and run the application bundles, and restore the system into its original state after application completion.

MySyS performs in the following way (Fig. 1):

0. When the deployment application receives the command to deploy the system, it injects a MySyS driver or kernel module bundle into the kernel of the host OS.

1. Driver virtualizes host OS on the fly by deploying MySyS multikernel hypervisor between hardware and host OS.

2. Hypervisor performs *intra-computer computations consolidation*, which is similar to the virtual machines consolidation in data centers. It collapses the host operating system to a smaller number of processor cores, maintaining the illusion for the host that it occupies all available CPU cores.

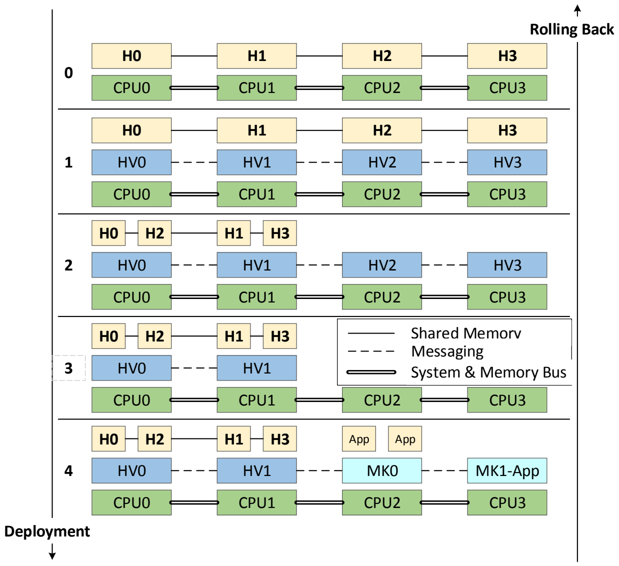

Figure 1: Multikernel deployment/rollback in MySyS. H - host CPU, HV - hypervisor, MK - multikernel node

3. Released CPU cores disconnect from the hypervisor network and go down. Moreover, the MySyS hypervisor reserves the requested amount of physical memory using the ballooning technique [60] and isolates this memory from the host OS. Finally, it reserves and isolates requested I/O capabilities (PMIO, MMIO, IRQs).

4. Hypervisor deploys requested configuration of multikernel bundle using previously reserved resources (CPU cores, memory, and I/O facilities). It configures OS node bundles and boots them on free CPU cores. New multikernel OS nodes connect to the network of hypervisor nodes and run without virtualization. Instead, they use virtual memory facilities to isolate application from the rest of the system.

5. Transformation of host OS into multikernel finished.

During the multikernel mode of operation, the hypervisor mediates communication between processes running on the host OS with applications deployed in the multikernel partition of the system. At the same time, it mediates access from the multikernel network to the services of the host OS. Finally, it provides administration facilities to the system administrator, allowing him to modify the multikernel network, redistribute resources between the host OS partition and multikernel partition, redeploy multikernel node bundles, and roll back the host system back.

Once the job of applications deployed in the multikernel partition is finished, the hypervisor rollbacks the system to its original state. It resets previously allocated cores and reboots them with hypervisor nodes connected to the hypervisor network. Then hypervisor unconsolidates the computations, returning the host OS to its initial deployment state. It also releases ballooned memory and returns to host OS I/O facilities. After that, it devirtualizes the host OS by removing hypervisor nodes from the system. Finally, the driver unloads, and the host OS stays in the original state without any sign showing that it was in a multikernel state previously.

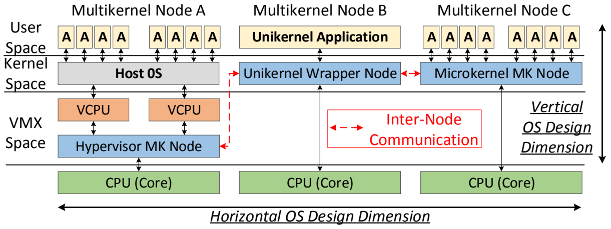

Figure 2: MySyS structure after deployment of application bundles (A – Application).

In contrast to many other multikernels, MySyS assumes neither strict specific structure (neither in space nor in time) nor homogeneity of the operating system's nodes. Each application bundle consists of a set of processes and a multikernel node (kernel) dedicated and specialized for their management and control and, after deployment, becomes a new multikernel node. The simplest form of such an application bundle is a single-process unikernel accompanied by a kernel wrapper. However, it can be a multi-process application managed by a dedicated microkernel node. Therefore, all kernels forming multikernel networks are diverse and application-specific, while the network is highly flexible and dynamic in space and time.

For instance, the hypervisor MK nodes serve to collaboratively and seamlessly virtualize and isolate the unmodified host operating system and consolidate it. It consists of top and bottom layers. The top layer encompasses one or more VCPUs and handles interaction between the host OS and hypervisor. The bottom layer manages the physical CPU and all interrupt sources (Local and IO APICs, PIT, PCI controller). Hypervisor nodes perform IPI and IRQ routing, reserve and isolate computer resources required for the application bundles, schedule VCPUs concurrency, and deploy bundles.

Unikernel wrapper is trivial. It sets up virtual address space to isolate the unikernel application, bypasses all resources required for its execution, and sets exception handlers to intercept its misbehavior. The current design assumes that the unikernel application (network service) will run in user space without the need to access kernel services.

A microkernel node is a shrunk second-generation microkernel. It performs intra-node scheduling, memory management, and IPC. It assumes a complex computational-intensive workload and, because of this, lacks IO management facilities.

There are only two functions common to all kernels participating in the MySyS multikernel network: inter-kernel communication, required for network integration into a single system, and proper isolation of workload applications.

7 DISCUSSION

MySyS employs a hypervisor solely for on-the-fly virtualization and isolation of the host OS and does not require virtualization for multikernel application bundles (but does not prohibit it). Instead, multikernel OS nodes can use more performant isolation based on virtual memory. Furthermore, MySyS does not pose any limitation or specific requirement on the multikernel node design except the usage of specified communication and configuration protocols. Multikernel nodes can be as simple as unikernel-like applications and as complex as the full-featured OS with a kernel and multiple applications and services running on top. Finally, in MySyS, the hypervisor controls and manages the allocation of resources only on the host OS side. Multikernel OS nodes can implement any protocol or policy for redistribution, isolation, and sharing of the resources allocated from the host. Hypervisor maintains SSI abstraction but only for transparent virtualization of the host OS and does not require or prohibit its implementation in other nodes of the multikernel network.

MySyS framework is mostly OS-agnostic. The only OS-dependent part is the wrapping driver, while the hypervisor and application bundles are independent of the host OS. Due to this, MySyS can be adapted to any OS that allows the injection of kernel modules (drivers) into kernel space.

In contrast to many recently proposed systems exploring kernel specialization but targeting exotic [40] or future hardware platforms [2, 28], the MySyS framework is designed and intended to run on commonly available multi- and many-core x86 processors and in collocation with widespread operating systems. However, it is easy to note that it can adapt to heterogeneous and cache-non-coherent hardware environments without significant investments.

7.1 Dimensions in OS design space

Our experience is that we should consider multikernel not as just another participant of the long-lasting debate "monolithic kernel or microkernel or exokernel or hypervisor" but as a new dimension in the OS design space. All previously existing approaches to OS design were invented in the era of single-core processors and implicitly assumed hierarchical vertical structuring of the OS, debating which part of functionality on what layer of the software stack and in what form should be placed. In contrast, multikernel assumes horizontal structuring of the OS as a network. Thus, we argue that there could be an open debate about multikernel network structure, configuration, and topology. However, in general vertical and horizontal dimensions in OS design space are ortogonal to each other (Fig. 2). For the same reason we cannot consider multikernel as a new layer in OS structure. In contrast, all available kernel designs can be adapted and integrated by multikernel OS.

7.2 Software complexity, requirements, and virtualization

In recent decades not only computer architecture experienced significant changes. Evergoing growth of software complexity fueled the rise of virtualization. However, in most cases, virtualization is used not for hardware emulation but as a facility simplifying the management, deployment, control, and running of complex applications – mega applications. Such mega applications consist of multiple applications/processes bundled, packaged, and isolated in a single entity accompanied by a set of required libraries, runtimes, policies, and kernel managing them while communicating with an external world through virtualized NIC interface. At the same time, growing diversity in application requirements and demands for performance force researchers to explore unikernel-over-hypervisor architectures where they face a need for extended, more capable, and feature-reach but still performant unikernels [48, 62]. However, MySyS solves the same task of deployment and running specialized complex applications, but in an alternative way.

7.3 Bluring the borders of OS notion

MySyS provides the original viewpoint on the system. In fact, it blurs and dissolves previously firm borders between notions of the operating system, kernel, driver, and application. For instance, the host OS view the entire MySyS package as a single application. But at the same time, the host OS is viewed by the deployed multikernel as a single service represented by a group of hypervisor nodes. One of our OS nodes unites the NIC driver, kernel, and application functionality bundled together into a single "application node" that the hypervisor deploys as a single entity. But at the same time, a microkernel-based node serves as a platform on which multiple processes execute concurrently.

8 CONCLUSIONS

Multikernel OS design was introduced more than ten years ago in reply to a significant shift in CPU architectures – the rapid spreading of multi-core processors. It has motivated multiple research projects exploring multikernel design. However, multikernel still fails to find a way to our computers. Meantime, multikernel can produce such valuable advantages as flexibility, fault tolerance, and full dynamism of software even on the current systems and can be adapted for practical use to solve actual problems now. We reconsidered the multikernel design principles and showed how to bridge the gap between multikernel and current computer systems. We hope that the proposed way will serve as an incentive for a smooth transition of the industry to novel architectures in hardware and software.

REFERENCES

[1] K. Adams and O. Agesen. A Comparison of Software and Hardware Techniques for X86 Virtualization. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XII, page 2–13, New York, NY, USA, 2006. Association for Computing Machinery.

[2] N. Asmussen, M. Völp, B. Nöthen, H. Härtig, and G. 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 '16, page 189–203. Association for Computing Machinery, 2016.

[3] A. Barbalace, B. Ravindran, and D. Katz. Popcorn: a replicated-kernel OS based on Linux . In *Proceedings of the 2014 Ottawa Linux Symposium*, OLS '14, 2014.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization . In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, page 164–177, New York, NY, USA, 2003. Association for Computing Machinery.

[5] A. Baumann. Hardware is the New Software . In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, page 132–137, New York, NY, USA, 2017. Association for Computing Machinery.

[6] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania. The Multikernel: A New OS Architecture for Scalable Multicore Systems . In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 29–44, New York, NY, USA, 2009. Association for Computing Machinery.

[7] A. Baumann, S. Peter, A. Schüpbach, A. Singhania, T. Roscoe, P. Barham, and R. Isaacs. Your Computer is Already a Distributed System. Why Isn't Your OS? In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09, page 12. USENIX Association, 2009.

[8] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 49–65. USENIX Association, 2014.

[9] A. Bhardwaj, C. Kulkarni, R. Achermann, I. Calciu, S. Kashyap, R. Stutsman, A. Tai, and G. Zellweger. NrOS: Effective Replication and Sharing in an Operating System. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 295–312. USENIX Association, jul 2021.

[10] D. Boggs, A. Baktha, J. Hawkins, D. T. Marr, J. A. Miller, P. Roussel, R. Singhal, B. Toll, and K. Venkatraman. The Microarchitecture of the Intel Pentium 4 Processor on 90nm Technology . *Intel Technology Journal*, 8(1):9–25, Feb. 2004.

[11] S. Borkar. Thousand Core Chips: A Technology Perspective. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, page 746–749. Association for Computing Machinery, 2007.

[12] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich. An Analysis of Linux Scalability to Many Cores . In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, page 1–16, USA, 2010. USENIX Association.

[13] R. Buyya, J. Broberg, and A. M. Goscinski. *Cloud Computing Principles and Paradigms.* Wiley Publishing, 2011.

[14] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Gener. Comput. Syst.*, 25(6):599–616, jun 2009.

[15] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: An Object-Oriented Approach to Non-Uniform Cluster Computing . *SIGPLAN Not.*, 40(10):519–538, Oct. 2005.

[16] J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska. Sharing and Protection in a Single-Address-Space Operating System. *ACM Trans. Comput. Syst.*, 12(4):271–307, nov 1994.

[17] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 08)*, San Francisco, CA, Apr. 2008. USENIX Association.

[18] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters . *Commun. ACM*, 51(1):107–113, Jan. 2008.

[19] A. Fedorova, M. Seltzer, C. Small, and D. Nussbaum. Performance of Multithreaded Chip Multiprocessors and Implications for Operating System Design. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, page 26, USA, 2005. USENIX Association.

[20] D. Geer. Industry Trends: Chip Makers Turn to Multicore Processors. *Computer*, 38(5):11–13, may 2005.

[21] B. Gerofi, R. Riesen, M. Takagi, T. Boku, K. Nakajima, Y. Ishikawa, and R. W. Wisniewski. Performance and Scalability of Lightweight Multikernel Based Operating Systems. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IPDPS'18, pages 116–125. IEEE, 2018.

[22] B. Gerofi, M. Takagi, and Y. Ishikawa. *IHK/McKernel* , volume Operating Systems for Supercomputers and High Performance Computing of *High-Performance Computing Series*, chapter 17, pages 291–306. Springer, Singapore, 2019.

[23] D. Greenley, J. Bauman, D. Chang, D. Chen, R. Eltejaein, P. Ferolito, P. Fu, R. Garner, D. Greenhill, H. Grewal, K. Holdbrook, B. Kim, L. Kohn, H. Kwan, M. Levitt, G. Maturana, D. Mrazek, C. Narasimhaiah, K. Normoyle, N. Parveen, P. Patel, A. Prabhu, M. Tremblay, M. Wong, L. Yang, K. Yarlagadda, R. Yu, R. Yung, and G. Zyner. Ultrasparc: The next generation superscalar 64-bit sparc. In *Proceedings of the 40th IEEE Computer Society International Conference*, COMPCON '95, page 442, USA, 1995. IEEE Computer Society.

[24] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *IEEE Micro*, 26(2):10–24, mar 2006.

[25] S. M. Hand. Self-Paging in the Nemesis Operating System. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 73–86. USENIX Association, 1999.

[26] G. Heiser, K. Elphinstone, S. Russell, and J. Vochteloo. Mungi: A Distributed Single-Address-Space Operating System. In *Proceedings of the 17th Australasian Computer Science Conference (ACSC)*, pages 271–80, Christchurch, New Zealand, Jan. 1994.

[27] J. Held, J. Bautista, and S. Koehl. From a Few Cores to Many: A Terascale Computing Research Overview. Technical report, Santa Clara, CA, USA, 2006.

[28] M. Hille, N. Asmussen, H. Härtig, and P. Bhatotia. A Heterogeneous Microkernel OS for Rack-Scale Systems. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*, APSys '20, page 50–58. Association for Computing Machinery, 2020.

[29] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson. A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, ISSCC '10,

pages 108–110. IEEE, 2010.

[30] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks . *SIGOPS Oper. Syst. Rev.*, 41(3):59–72, Mar. 2007.

[31] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, mar 2003.

[32] T. Killalea. Meet the Virts: Virtualization Technology Isn't New, but It Has Matured a Lot over the Past 30 Years. *Queue*, 6(1):14–18, jan 2008.

[33] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel . In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 207–220. Association for Computing Machinery, 2009.

[34] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, SP '19, page 1–19. IEEE, 2019.

[35] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ISCA '04, page 64. IEEE Computer Society, 2004.

[36] Y. Li, M. Danish, and R. West. Quest-V: A Virtualized Multikernel for High-Confidence Systems . *CoRR*, abs/1112.5136, 2011.

[37] J. Liedtke. On Micro-Kernel Construction . In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, page 237–250, New York, NY, USA, 1995. Association for Computing Machinery.

[38] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, Baltimore, MD, Aug. 2018. USENIX Association.

[39] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library Operating Systems for the Cloud . *SIGARCH Comput. Archit. News*, 41(1):461–472, Mar. 2013.

[40] S. Mallon, V. Gramoli, and G. Jourjon. *DLibOS: Performance and Protection with a Network-on-Chip*, page 737–750. Association for Computing Machinery, New York, NY, USA, 2018.

[41] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici. My VM is Lighter (and Safer) than Your Container . In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 218–233, New York, NY, USA, 2017. Association for Computing Machinery.

[42] T. G. Mattson, R. Van der Wijngaart, and M. Frumkin. Programming the Intel 80-Core Network-on-a-Chip Terascale Processor. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08. IEEE Press, 2008.

[43] R. McDougall and J. Laudon. Multi-Core Microprocessors Are Here. *login Usenix Mag.*, 31(5):32–39, october 2006.

[44] S. Mirapuri, M. Woodacre, and N. Vasseghi. The Mips R4000 Processor. *IEEE Micro*, 12(2):10–22, mar 1992.

[45] S. J. Mullender, G. van Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren. Amoeba: A Distributed Operating System for the 1990s . *Computer*, 23(5):44–53, May 1990.

[46] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The Case for a Single-Chip Multiprocessor. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VII, page 2–11. Association for Computing Machinery, 1996.

[47] S. Peter, J. Li, I. Zhang, D. R. K. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe. Arrakis: The Operating System Is the Control Plane. *ACM Trans. Comput. Syst.*, 33(4), nov 2015.

[48] A. Raza, T. Unger, M. Boyd, E. Munson, P. Sohal, U. Drepper, R. Jones, D. B. de Oliveira, L. Woodman, R. Mancuso, J. Appavoo, and O. Krieger. Integrating Unikernel Optimizations in a General Purpose OS, 2022.

[49] J. S. Robin and C. E. Irvine. Analysis of the intel pentium's ability to support a secure virtual machine monitor. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, page 10. USENIX Association, 2000.

[50] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: A Many-Core X86 Architecture for Visual Computing. *ACM Trans. Graph.*, 27(3):1–15, aug 2008.

[51] S. Siddha. Multi-Core and Linux Kernel. Technical report, San Francisco, CA, USA, 2007.

[52] S. Siddha, V. Pallipadi, and A. Mallick. Process Scheduling Challenges in the Era of Multi-core Processors . *Intel Technology Journal*, 11(4):361–369, Nov. 2007.

[53] R. L. Sites. Alpha AXP Architecture. *Commun. ACM*, 36(2):33–44, feb 1993.

[54] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors . In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, page 275–287, New York, NY, USA, 2007. Association for Computing Machinery.

[55] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, page 1–14, USA, 2001. USENIX Association.

[56] D. Tam, R. Azimi, and M. Stumm. Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, page 47–58, New York, NY, USA, 2007. Association for Computing Machinery.

[57] A. S. Tanenbaum and R. Van Renesse. Distributed Operating Systems . *ACM Comput. Surv.*, 17(4):419–470, Dec. 1985.

[58] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith. Intel Virtualization Technology. *Computer*, 38(5):48–56, may 2005.

[59] M. von Tessin. The Clustered Multikernel: An Approach to Formal Verification of Multiprocessor OS Kernels . In *2nd Workshop on Systems for Future Multi-core Architectures*, pages 1–6, Bern, Switzerland, Apr. 2012.

[60] C. A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, dec 2003.

[61] L. Wang, G. von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu. Cloud Computing: a Perspective Study. *New Generation Computing*, 28(2):137–146, apr 2010.

[62] Y. Zhang, J. Crowcroft, D. Li, C. Zhang, H. Li, Y. Wang, K. Yu, Y. Xiong, and G. Chen. KylinX: A dynamic library operating system for simplified and efficient cloud virtualization. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 173–186, Boston, MA, July 2018. USENIX Association.