Software-Defined "Hardware" Infrastructures: A Survey on Enabling Technologies and Open Research Directions

Amir Roozbeh, João Soares, Gerald Q. Maguire Jr., Fetahi Wuhib, Chakri Padala, Mozhgan Mahloo, Daniel Turull, Vinay Yadhav, Dejan Kostić

***Abstract*— This paper provides an overview of Software-Defined "Hardware" Infrastructures (SDHI). SDHI builds upon the concept of hardware (HW) resource disaggregation. HW resource disaggregation breaks today's physical server-oriented model where the use of a physical resource (e.g., processor or memory) is constrained to a physical server's chassis. SDHI extends the definition of Software-Defined Infrastructures (SDI) and brings greater modularity, flexibility, and extensibility to cloud infrastructures, thus allowing cloud operators to employ resources more efficiently and allowing applications not to be bounded by the physical infrastructure's layout. This paper aims to be an initial introduction to SDHI and its associated technological advancements. The paper starts with an overview of the cloud domain and puts into perspective some of the most prominent efforts in the area. Then, it presents a set of differentiating use-cases that SDHI enables. Next, we state the fundamentals behind SDI and SDHI, and elaborate why SDHI is of great interest today. Moreover, it provides an overview of the functional architecture of a cloud built on SDHI, exploring how the impact of this transformation goes far beyond the cloud infrastructure level in its impact on platforms, execution environments, and applications. Finally, an in-depth assessment is made of the technologies behind SDHI, the impact of these technologies, and the associated challenges and potential future directions of SDHI.**

***Index Terms*—CR-Software-Defined Infrastructure, Resource Disaggregation, Cloud Infrastructure, Rack-scale, Hyperscale Computing, Disaggregated DC**

I. Introduction

Cloud computing continues to be an emerging field that accelerates digital transformation and innovations in many other technology fields, such as mobile communication [1–7], robotics [8], and Internet of Things (IoT) [9]. Over the years, the information technology (IT) industry has been continuously searching for ways to improve the efficiency of cloud infrastructures (specifically data centers (DCs)), regardless of their size). By improved efficiency we mean, among other aspects, reducing resource wastage and faster infrastructure adaptation. Examples of such improvements are the introduction of server virtualization technologies, such as hypervisor & container technologies [10] and the move towards Software-Defined Infrastructures (SDI). With SDI the entire DC's infrastructure is abstracted, becoming software-defined and dynamically programmable [11].

The abstraction introduced by SDI enabled workloads to be transparently moved within and across cloud infrastructures. For example, this allows applications to quickly scale up/down when they could not do so in the particular infrastructure node where they were initially running. Additionally, these technologies allow infrastructure operators to optimize the infrastructure's efficiency (e.g., reducing resource wastage and energy consumption) and facilitating maintenance (e.g., replacement of servers) by moving workloads around.

In the effort to continue increasing application and infrastructure efficiency, additional approaches, such as microservices [12] and serverless computing [13] have been introduced. Microservices increase modularity by decomposing today's traditional applications into smaller service units, allowing more fine-grained control. In serverless computing, server management and capacity planning aspects are abstracted away and completely hidden from the developer or DC operator. The function-as-a-Service concept [14] is one way to achieve a serverless architecture, while bringing an increased level of granularity and providing more fine-grained control. All of these technologies are primarily based on software mechanisms deployed and running on top of the cloud infrastructure in order to hide and deal with the limitations of this infrastructure.

Current cloud infrastructure architectures build upon a server-oriented model, hence they are constrained by server boundaries and the servers' physical hardware configuration. While SDI and other software approaches continue to improve the efficiency of such infrastructures, industry has realized that to achieve further improvements in efficiency requires changes in some of the today's primary infrastructure principles. This has led to the concept of hardware (HW) resource disaggregation and Software-Defined "Hardware" Infrastructures (SDHI). Here the term "hardware" is used to emphasize the fact that the concept is not simply applying traditional virtualization or other types of software approaches on top of legacy infrastructures.

SDHI requires the cloud infrastructure to be re-architected in fundamental ways. SDHI changes this model by introducing resource disaggregation into the infrastructure. Resource disaggregation breaks the current physical server boundaries and considers physical resources as individual and modular components. These resources are then controlled through software, allowing the composition of a logical server that replaces a motherboard. The resulting logical server interconnects a set of different resources over which an operating system (OS) runs on an interconnected set of resources allocated from pools of resources.

This new approach extends the definition of SDI and brings greater modularity, agility, flexibility, and extensibility of the cloud infrastructure. In SDHI a unified software layer controls the (logical) interconnection between different resources that realize logical server systems. While some of the technologies required for SDHI are already in place today [15], there is still a journey ahead to realize the full potential of SDHI.

The contributions of this paper are numerous. One of the key contributions of this survey is to present a consolidated in-depth study of the enabling technologies and ongoing efforts regarding SDHI. Several *ad hoc* technological advancements in SDHI have already been made, while others are still ongoing. However, there is no introduction to these in the literature. However, there are technology surveys available that look, individually, into technologies that relate to SDHI, such as virtualization technologies [16] or SDN [17]. This paper aims to be the first introduction to SDHI related technology advancements and existing surveys. Moreover, the paper examines how a shift in hardware and the concept of HW resource disaggregation impacts SDI and the entire cloud architecture, from the HW level up to the application level. Furthermore, the paper discusses previously identified challenges, surveys existing works, and identifies some unforeseen challenges and open research directions. Finally, it provides a set of use-cases that SDHI will unlock. We believe our approach makes this paper a valuable resource for both the research community and practitioners seeking to understand the complexities of SDHI and to help them focus on the key challenges of this concept.

The remainder of this paper is organized as follows. Section II briefly describes traditional cloud infrastructure architectures and elaborates the principle of SDI and its current level of maturity. Moreover, it elaborates the SDHI terminology and discusses the drivers that motivated the realization of SDHI. Section III discusses the market demand for SDHI and describes some of the technical use-case opportunities it will unlock. Section IV examines the key architectural and functional components of a cloud infrastructure ecosystem concerning SDHI. Next, Section V gives a technology overview and describes a range of ongoing efforts toward realizing SDHI. Additionally, it identifies technical challenges and open research directions regarding SDHI technology. Section VI briefly discusses related research initiatives, and refers to a set of tools that can be used to further explore the area of SDHI. Finally, Section VII presents a summary of lessons learned and a set of concluding remarks.

II. Software Defined Infrastructure Background

Today cloud providers are utilizing virtualized and SDI environments to achieve a higher degree of cost reduction, agility, optimization, programmability of the infrastructure, and automation. Despite all these measures, SDI is limited and tightly coupled to the traditional server architecture. Servers in today's cloud infrastructures are pre-defined at the factory where hardware resources are assembled within a chassis and cannot be used directly beyond that chassis. HW resource disaggregation expands the boundary of SDI by breaking this tight coupling and allowing resources to be handled as independent resources regardless of the physical chassis they are placed in.

A. Server-based SDI

SDI was a response to the demand for greater flexibility, agility, and optimization. As shown in Fig. 1, several other terms with a tight connection and similarity with SDI terminology have also been used in the literature, such as: Software-Defined Environment (SDE), Software-Defined DC (SDDC), and Software-Defined Cloud (SDCloud) (see Table I). By SDI we mean the entire DC's infrastructure is software-defined and dynamically programmable. SDI can be realized by different building blocks including Software-Defined Computing (SDC) and virtualization, Software-Defined Networking (SDN) and virtual network function (VNF), and Software-Defined Storage (SDS) (see Table II).

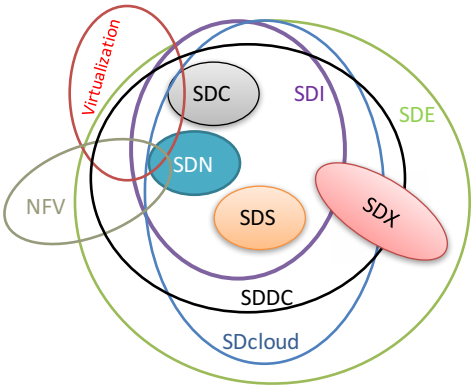

Fig. 1. Taxonomy of software defined environments.

The current cloud infrastructure architecture and SDI are based upon a physical server-oriented model, where the infrastructure (i.e., a DC) is composed of pools of servers. These servers typically consist of a physical circuit board predefined and populated at the factory with specific resources, such as central processing units (CPUs), co-processors, memory, and networking subsystems. As shown in Fig. 2, these resources are interconnected using specialized communication channels, such as Peripheral Component Interconnect Express (PCIe), Double Data

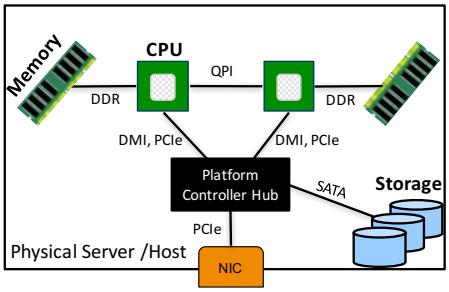

Fig. 2. Simplified physical server component architecture.

Rate (DDR), SerialAT Attachment (SATA), Quick Path Interconnect (QPI), and Direct Media Interface (DMI). Communication between server components is limited to the server's chassis and has strict requirements on latency and bandwidth to enable the system to work as an integrated system. Table III summarizes these requirements. Servers are interconnected via their network interface (generally referred to as a NIC) via a DC-wide network, as shown in the left side of Fig. 3.

TABLE I
TERMINOLOGIES SIMILAR TO SDI TERMINOLOGY.

Terminology	Description
SDE [11]	All the components in an environment (e.g. IT infrastructure, power systems, and cooling systems) are software-defined and dynamically programmable.
SDI [11]	The entire DC IT infrastructure is software defined and dynamically programmable.
SDCloud [18, 19]	The process of cloud configuration automated by extending virtualization to all of the Cloud infrastructure's physical resources in a DC.
SDDC [20]	The software-defined concepts is integrated into all DC's main blocks, such as network, storage, compute and security.

TABLE II
TERMINOLOGIES INVOLVED IN SDI.

Terminology	Description
SDN [17]	Separates the network control planes from data planes and physical network entities to improve programmability, efficiency, and extensibility of network.
SDS [21]	Separates the control planes from the data plane of a storage system enabling heterogeneous storage to respond dynamically to changing workload demands.
SDC [11]	Originated from the computing environment in which the computing functions are virtualized and managed as virtual machines through a central interface as one element.
Virtualization [16, 17]	A software abstraction layer (i.e., hypervisor) emulates the behavior of physical entities in software. This term has been used in two different contexts in the literature, i.e., network virtualization and server virtualization. See Section V-D2.
VNF [22, 23]	Relocates network functions from dedicated network appliance to commodity servers.

TABLE III
APPROXIMATE COMMUNICATION REQUIREMENTS BETWEEN DIFFERENT RESOURCES WITHIN A SERVER. THESE VALUE DIFFER FOR DIFFERENT HARDWARE [24, 25].

Communication type	Delay (ns)	$Bandwidth(Gbps)$
CPU - CPU	≈ 10	500
Memory - CPU	≈ 20	500
CPU - 10G NIC	$> 10^3$	10
CPU - SSD disk	$> 10^4$	5
CPU - HDD disk	$> 10^6$	1

Until recently, networking technologies were unable to overcome the need to place the different components very close in terms of physical proximity. This resulted in fixed server configurations, making adaptation to workloads extremely difficult. Furthermore, infrastructure lifecycle management is tightly bound to the lifecycle of a server chassis. This causes problems for providers who wish to upgrade part of their infrastructure, especially since the resources that compose a server have different lifecycles. For example, if an infrastructure provider wants to upgrade or increase its compute capacity by adding new memory or processor technology, in most cases this means replacing the entire server.

Another key challenge in today's infrastructures is low resource utilization [26, 27], when the resources the infrastructure operator has paid for are not utilized to their full capacity. One reason for this is resource stranding [28], because of resources left from fragmentation due to the mismatch between application requirements and the fixed configuration of each server. For example, a CPU-intensive application can exhaust a server's computational resources while stranding other resources, such as memory and storage. To cover peak demands infrastructure providers must over-provision. This leads to large aggregated amounts of stranded resources and hence low utilization.

In order to improve resource utilization, infrastructure providers employ advanced resource scheduling techniques, in an attempt to solve what it is in most cases an NP-hard optimization problem. Moreover, they rely upon virtualization technologies, such as containers [29, 30] and hypervisors [31]

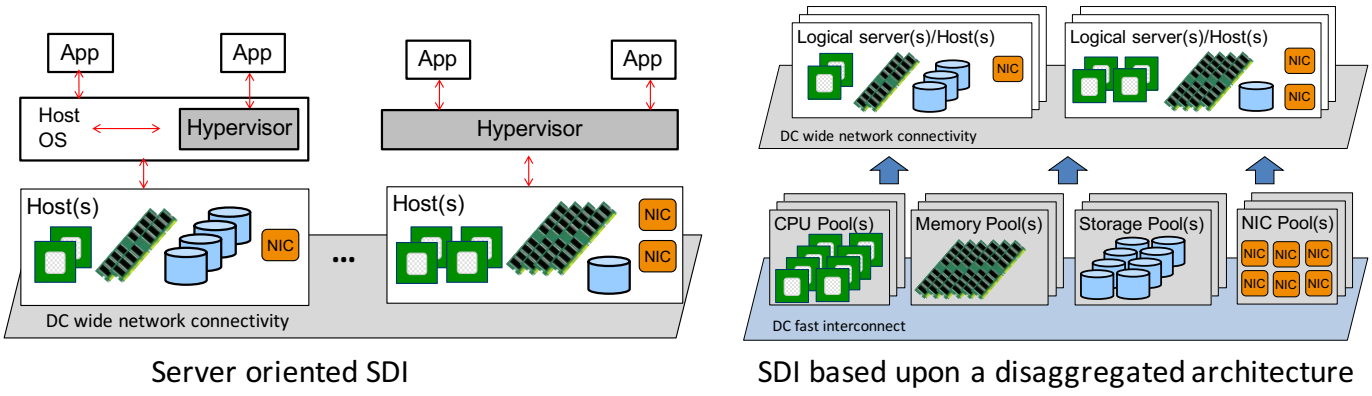

Fig. 3. Server-oriented SDI vs. SDI based upon resource disaggregation

(such as Xen [32], VMware [33], and KVM [34]), to implement server consolidation, multi-tenancy, and resource sharing. These techniques allow infrastructure providers to reduce their costs while providing dynamic scaling to their customers. In general, hypervisor virtualization can be classified into two types: the hypervisor runs directly on the hardware or the hypervisor runs on a host's OS - see the left sides of Fig. 3. However, hypervisor virtualization comes at the cost of extra overhead, as it interleaves execution of the applications and the execution of the hypervisor/host OS. Virtualization adds different amounts of overhead based on the type of virtualization and other competing workloads running on the same physical server [35–37]. In contrast, container-based virtualization has some benefits compared to hypervisor virtualization, especially in terms of performance. However, it is still subject to overheads and is highly dependent on the server's OS.

Despite all these measures, infrastructures still operate at low utilization [27] as today's approaches cannot overcome the physical boundaries of a server's chassis.

B. Software-Defined "Hardware" Infrastructure (SDHI)

SDHI brings a novel approach to cloud infrastructures by introducing resource disaggregation based on software-defined principles wherein all resources are pooled and managed by software. Resource disaggregation breaks the traditional physical server architecture boundaries and considers hardware resources as individual and modular components. An ultimate disaggregation scenario has each of the different resources organized in independent pools of computing units, memory units, storage units, network interfaces, and other resources (e.g., accelerators, such as Graphics Processing Unit (GPU)).

SDHI breaks the one-to-one mapping between a physical server's chassis and a host by decoupling the software and management components from the underlying hardware. This makes the infrastructure more programmable. When an SDHI is initially deployed, there is no notion of host/server systems, only knowledge of the different hardware components yet to be assigned and programmed. The cloud infrastructure operator then utilizes an application programing interface (API) to create specific host/server systems, i.e., logical servers or logical hosts.

These principles bring a high level of fluidity, modularity, and flexibility to the cloud infrastructure. In this way, it is possible to dynamically establish and adapt the configuration of logical servers to precisely match the needs of particular workloads by on-demand selection and configuration of (physical) resources from different resource pools. This requires the necessary physical resources to be interconnected via high-speed very low latency communication to handle the physical separation between the different physical resources.

The interconnection fabric plays a pivotal role within a SDHI architecture, being both an enabler and blocking factor. Networking can be realized by two separate networks with different requirements: a fast interconnect fabric between individual hardware components and a DC-wide interconnection via NICs. The fast interconnect fabric interconnects a set of hardware resources from which logical hosts can be realized as an integrated system. As a result, the fast interconnect fabric deals with the traffic *within* logical hosts. In contrast, in server-based SDI model, this interconnection traffic was restricted to a physical server's motherboard. This fast interconnect fabric should support strict latency and bandwidth requirements [24], while providing substantial resiliency, low cost, and low power consumption. The DC-wide interconnection enables logical hosts to communicate with the external environment (e.g., the Internet or other logical hosts) via the logical host's NICs. This traffic corresponds to the network traffic that exists in today's DC. The right side of Fig. 3 illustrates SDHI realized as a set of resource pools.

Ultimately, with SDHI, each hardware resource can serve multiple hosts, and a single host can consume resources from multiple hardware resources, thus allowing operators to optimize their resources.

Researchers describe somewhat different levels and types of resource disaggregation [38]. In the majority of cases, disaggregation is restricted to a rack (i.e., logical hosts can only be composed within a rack, therefore the term "rack-scale"). This restriction is mostly due to network limitations. However, we believe these limitations will be overcome in the long run. Moreover, in some cases, these systems are composed of microservers (each with a heterogeneous set of resources) interconnected through high-bandwidth low-latency networks. In other cases, they consist of different pools of resources (i.e., each pool having one kind of resource).

The concept of SDHI captured the interest of both industry and research community. One tangible realization is Intel's Rack Scale Design (RSD) [39, 40]. RSD is one of the pillars of the Open Compute Project (OCP) [41], an open source initiative that aims to scale computing infrastructures in the most efficient and economical way. Moreover, RSD is part of the foundation of Ericsson's Hyperscale Data System (HDS) 8000 [15]. Other industrial efforts in the area include Memory-Driven Computing [42], Rack-Scale Computing [43], and others [44–46]. Moreover, research initiatives trying to advance the state-of-the-art are also taking place [47–52]. We further elaborate on these in the following sections.

At this point, it is important to highlight that currently there is not (yet) a fully SDHI environment. Most of the systems only support partial disaggregation (e.g., storage [15, 41]) and are limited to the rack level. Base solutions for the overall system management entities are available (e.g., Intel RSD [39]). However, technology is evolving, and with time further disaggregation will occur (i.e., memory, network interfaces, and others), putting increased and more complex requirements on the management systems.

III. New Opportunities and Applications

This section discusses why the market is demanding SDHI, and then elaborate on some of the technical use-case and workload deployment opportunities that are made possible by SDHI. These use cases illustrate the possibility of innovations in the way we run application and workloads in DCs.

A. Why Now? Market Demand

The rapid digitalization of industries, combined with the concept of IoT [53] and the ongoing transformation of companies through the use of cloud solutions, are just a few of the factors that have driven the increase in IT capacity [54]. In consequence, global spending on DC systems is growing [55] and there have been efforts to move toward SDI driven by economic and pricing theories for cloud resource provisioning, cloud management, and cloud networking [56, 57] in order to achieve sustainable profit, cost reduction, and flexibility. However, because the current ratio between IT capacity and its related cost is still high, it is not easy to deliver the desired capacity by using current SDI technologies simply with increased investment. In addition to financial barriers, the increasing complexity of new services, such as their high dynamicity, brings new challenges for future cloud deployments. These facts have led the IT community to search for ways to scale data center (DC) infrastructures beyond the cost and capacity limitations of today's DC architecture [58]. The first step is to rethink the DC architecture and make it more modular, flexible, and smart. The SDHI concept is driving this architecture change.

SDHI promises to bring new functional and business opportunities by extending the boundary of current SDI. It promises to be a step towards reduced Total Cost of Ownership (TCO) [59, 60]. For example, hardware utilization in current DCs is under 50 percent [61]. Until recently this was acceptable, but that is no longer the case. The market is too aggressive to ignore such wastage of resources. Hence, everything that can significantly affect the operational performance of the cloud infrastructure matters. Consequently, economic efficiency is vital for cloud owners of all sizes. The cost benefits of SDHI can be divided into business opportunities and TCO reduction [62] (as described below).

Business opportunities: Combining resource disaggregation and software-defined capabilities makes it less expensive and faster to expand the infrastructure to follow the exponential growth in both data and demand for IT services. Moreover, it offers the possibility to right-size the infrastructure based on the actual market [63] and customer demand, while avoiding over-dimensioning or losing customers due to insufficient capacity. New capacity can be added by either upgrading current components with more powerful ones (e.g., upgrading CPUs to ones with higher clock frequencies or larger core counts) or simply adding new chassis of the specific components which are the bottlenecks (e.g., if there is an increase in the amount of data, expanding the storage tier and memory tier, without needing to purchase new and costly CPUs). Moreover, the inherent flexibility and dynamicity of this type of infrastructure facilitates the adaptation of resources to fit any type of workload or application, further reducing the risk of investments in the infrastructure. Improved efficiency and finer granularity in creating logical servers bring new revenue streams for the cloud infrastructure operator. This makes it possible for these operators to act as a service provider by using these extra resources to offer new services to their customers at low prices.

TCO reduction: Low resource utilization is one of the primary reasons for the high cost of computing units [64]. It is widely recognized that cloud infrastructures run at very low levels of resource utilization due to their physical silos, hence resources are being wasted. Resource disaggregation provides a greater degree of sharing, thus enabling higher utilization. Higher utilization means that the same workload can be served with less hardware [65], hence reducing capital expenditures (CAPEX). However, cost savings go beyond CAPEX and can impact operational expenditures (OPEX).

SDHI's TCO related aspects have been thoroughly explored in [60] via a framework to assess the cloud infrastructure's economic efficiency. This work presents a thorough cost comparison of deploying an SDHI architecture versus deploying a server-based SDI architecture in DCs. The study considers all the major cost categories incurred during the DC's lifetime with regard to CAPEX and OPEX (i.e., from the deployment phase, when a huge upfront investment is required, to all costs related to each operational process). Results show that in the presence of heterogeneous workloads, having a DC based on SDHI brings high savings (of more than 40% depending on the applications) compared to server-based SDI. Moreover, lifecycle management cost was one of the main differentiators between SDHI and SDI.

Electricity is one of the limiting factors for deploying DCs. One of the direct consequences of increased resource utilization which comes from SDHI is reduction in the DC's power consumption. As fewer resources have to be turned on for the same workload, the power consumption per unit of workload is lower. Using more energy efficient algorithms for workload placement, which is managed by SDHI, can further reduce energy consumption, leading to a greener DC. Reduction in power consumption, leads to less heat dissipation, hence less cooling power, which contributes to increased power savings compared to DCs operated with server-based SDI [66]. Also, the architecture of the SDHI DCs, including the way cooling system is designed, can have a large impact on the energy consumption of a DC. For example, [67] proposes a novel energy-efficient architecture for software-defined DC infrastructures, were the total power consumption is reduced by 27% in comparison to a DC with a conventional architecture. Moreover, the capabilities of Smart Grids technologies [68–70] should be exploited within DCs based on SDHI to further save energy.

B. Technical Applications

1) Adaptive Server Realization: Ideally, there should be a perfect match between the workload being executed and the logical server's resources in an SDHI to eliminate resource stranding, thus gaining the most from this new paradigm.

With SDHI, the logical server and the platforms and applications it hosts (the service domain) can interact with the different operational layers (see Section IV) by sending requests and receiving hints and feedback. Tight interaction between the service and operation domains presents an

opportunity for dynamic configuration and fine-grain resource (re)scheduling that can increase resource utilization and system efficiency. Fig. 4 illustrates some scenarios where a logical server could benefit from dynamic adaptation.

Fig. 4(a) illustrates a scenario where the execution environment scheduler decides which job (i.e., part of a given workload) will be executed by which CPU, hence this scheduler can provide feedback to the management stack (step 1). This feedback can be used to optimize the configuration of the logical server. For example, the network can be reconfigured such that the part of the known workload will be executed in a particular logical server's CPU and hence will be allocated memory close to this CPU (step 2). Later the operations domain can be notified of either: (1) the logical server requiring more resources - Fig. 4 (b) - step 1 or (2) a set of resources may be released - Fig. 4(c) - step 1. In these cases, the operations domain dynamically re-configures the different resources and DC interconnection fabric to adapt the logical server to its new configuration (Fig. 4 (b) and (c) - step 2). Minimizing the time required for realizing these dynamic adaptations is a major factor in achieving high performance and efficiency.

Consistent monitoring of the application and using feedback from the execution layer enables the operations domain to identify deviations in the behavior of a given workload from the predicted behavior. A key question is to identify how frequently reconfiguration and rescheduling should take place in order to increase the efficiency of logical servers while considering the time required for re-configuring and re-scheduling the workload, the network conditions, and the availability of the necessary physical resources.

2) The Big Modular Machine: Today application designers have to deal with the resource limitations of a single server. This is particularly true for those applications that store and handle a large volume of data, thus requiring a large amount of memory. SDHI can solve this problem by forming "very" big machines that are composed by interconnecting a large number of CPUs, memory, and storage.

Applications (such as in-memory databases) that today rely on being distributed over a set of servers to achieve scale can potentially be deployed on a single big machine provisioned through SDHI. This will reduce the complexity of the application by eliminating the synchronization overheads, load balancing, and failure handling that arise due to scaling by distribution. This approach can also help improve the performance of the application by removing the communication overhead between the servers, as all the data and processing are resident within the single big machine. High-performance Computing (HPC) type of applications, real-time analytics, and in-memory software such as SAP-HANA (which today can be configured to operate on 8 cores and 6 TB of RAM) can potentially be scaled up to operate on a much larger number of CPUs and much larger amounts of memory. This scaling up is needed to cater for large-scale data handling in the future. Legacy applications (such as operations support systems and business support systems (OSS & BSS) were not designed to scale out but need to handle the demands of the future. However, these would benefit from being deployed on logically large machines, thus eliminating the need for scaling up individual processors.

3) Highly Available Server: High Availability (HA) is an important concept in building a robust service. Today ensuring HA of hardware components running a service requires full redundancy of the service on another server(s). In other words, the granularity is that of a server. This is not a cost-effective solution and the recovery time is large. Moreover, the application needs to be aware of the HA setup.

SDHI introduces new failure points, as the boundary of server extends beyond a single physical blade. Thus there are scopes for research in HA and fault resiliency models at HW and software levels. Nevertheless, in SDHI, HA can be provided on a component level rather than a server level. Logical servers composed using SDHI can include redundant components that can quickly take over from components that fail. This means that the failure of a single component in an SDHI composed logical server need not result in the failure of the entire logical server, hence the logical server can continue to function due to the redundant components. Component

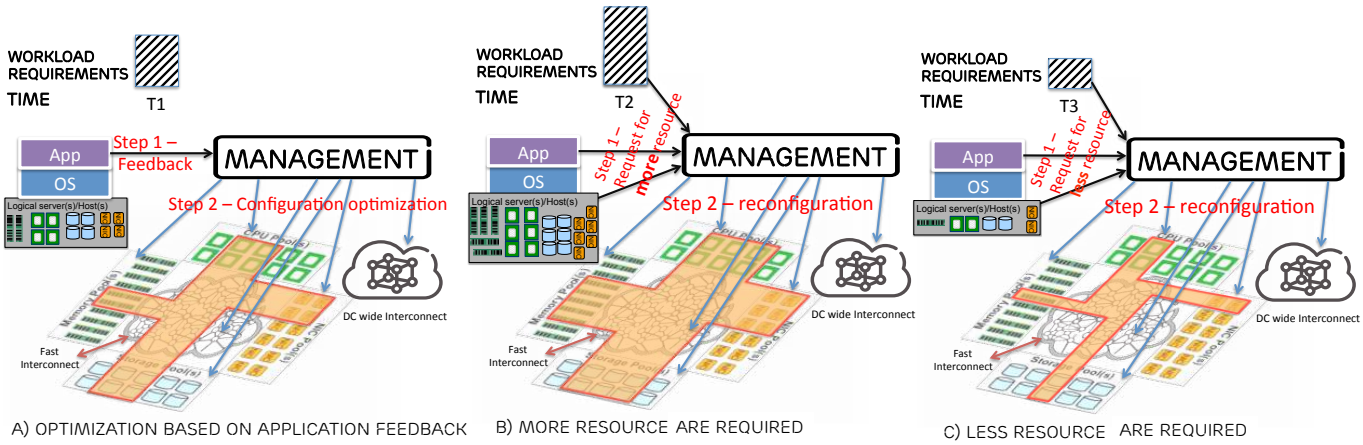

Fig. 4. Adaptive and dynamic logical server realization: (a) shows the configuration optimization based on application feedback, (b) shows what happens when a logical server requires more resources, and (c) shows what happens later when the logical server requires fewer resources.

level granularity combined with component health monitoring enables the addition of redundant components only of the type that are failing instead of allocating multiples of an entire redundant server as is done today. An SDHI based logical server can be made robust against failure of CPU, memory, disk, or NICs without incurring a long downtime, resulting in the HA mechanism being transparent to the applications, hence reducing their complexity.

4) In-memory communication and data sharing: A lot of network traffic that traverses network elements (such as switches in a DC) is between applications running on different servers. Today, in order to achieve scale, applications are composed of different components that can be run in a distributed fashion on different servers, but this may lead to an exchange of a large volume of data between the servers within a DC (or even between different DCs).

In an SDHI, logical servers can potentially share a portion of memory and use that memory as a communication channel. This mode of communicating via shared memory can reduce cost by reducing the number or capacity of NICs and switches in the DC (at the cost of using the interconnection fabric). Moreover, it can reduce latency for the communication between applications.

Today big data frameworks (such as Hadoop) need to keep data in a distributed fashion spanning a large number of servers [71]. A single large file can be decomposed into smaller blocks and stored on different servers. Accessing a part of the file from a server might require network communication to fetch the block of that file from a different server, which increases latency and hence negatively impacts performance. The flexibility of SDHI in assigning the resources to logical servers facilitates attaching a block of storage containing the required section of a file to a logical server which wants to operate on the data thus avoiding transfer of the data over the network. Another advantage is the possibility to dynamically compose a logical server for processing data around the storage component which holds the data. In a Map-Reduce use case, logical servers can be composed of storage components which hold the data required for the map operation. Once the map operation is completed and the intermediate data is written to the storage, new logical servers can be composed of those storage components holding the intermediate results for the reduce operation thus reducing the need to shuttle data between servers as it is traditionally done.

IV. DC ARCHITECTURE BASED ON SDHI

HW disaggregation and the associated concept of SDHI are rather disruptive technologies whose effects are far reaching — from HW design and manufacturing to how applications are developed for disaggregated HW systems. To understand the impact of HW disaggregation on the architecture of DCs, we start with common cloud service models and their associated stakeholders. Fig. 5 shows these stakeholders (in bold) in a generic DC deployment that provides all of the traditional "X" as a Service (XaaS) services. In this deployment, end users consume applications from software providers (i.e., Software as a Service (SaaS)). Application providers develop

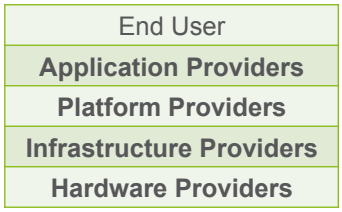

Fig. 5. Key stakeholders in a deployment of a cloud-type DC.

(or obtain developed) applications for a software platform that is operated and maintained by platform providers (i.e., Platform as a Service (PaaS)). Platform providers deploy their platform software on the (virtualized) infrastructure provided as a service by infrastructure providers (i.e., Infrastructure as a Service (IaaS)). Infrastructure providers, in turn, consume the services of HW providers (e.g., HW purchasing/leasing, colocation, and other utilities & facilities) to provision the HW infrastructure on which they provide their service.

A functional architecture of a DC system built on disaggregated HW resources can be defined with the above stakeholders in mind. Fig. 6 shows one such architecture that has a management layer that corresponds to each stakeholder. In addition to these layers, the functional blocks of the architecture are organized as *service-domain functions* (on the left) and *operations-domain functions* (on the right). Service domain functions relate to the "payload" (from the viewpoint of the DC), including the applications that are used by the end users of the cloud, the platform-level software (i.e., OS and supporting services needed to run the application), and HW resources used to execute the software. The operations domain functions, on the other hand, relate to the management functions that make the DC work, starting from those at the hardware level that allow resources to be split and composed into an SDHI, to those at the application level that manage the orchestration and deployment of application components. Management functions implemented by several existing cloud management systems, such as OpenStack [72], Ubuntu Metal as a Service (MaaS) [73], and Kubernetes [74] are examples of operations domain functions at the various layers of the architecture. However, beyond these traditional functions there are new server functions, such as those described in Section

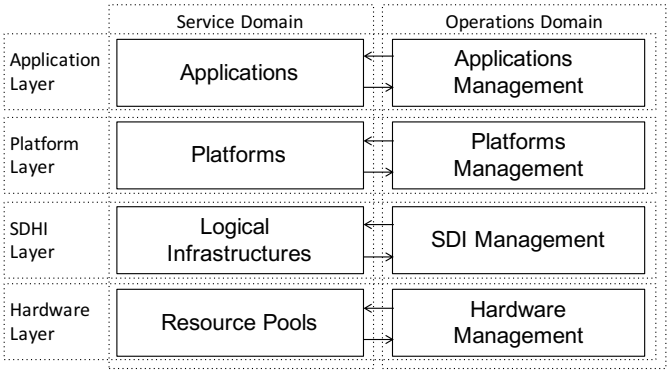

Fig. 6. High-level architecture for a DC built from disaggregated HW resources.

IV-A that are specifically needed to realize SDHI systems.

A. Hardware Layer

This layer includes the physical resources and their associated interconnects/fabrics from which the SDHI can be realized (in the service domain), as well as the HW-near functions that enable disaggregated HW to be pooled, aggregated, and composed into a software-defined logical infrastructure (in the operations domain). In the service domain, this architecture does not make specific assumptions with regard to the level of HW disaggregation or the specific functionalities available at the HW-level. However, it does assume that the available HW is composable in the sense that it is possible to dynamically compose and recompose logical infrastructures from the HW resources and HW resources are organized into interconnected *resource pools*.

The operations domain functions at this layer include traditional server management functions that are implemented in microcontrollers, such as IPMI [75], HP's ILO [76], and Dell's iDRAC [77]. However, there are a new set of management functions (such as the composability service in Redfish [78] - see section V-C1) which may present an important management overhead when compared to the management of traditional server-based systems.

In the service domain, performance implications need to be taken into account, namely due to additional networking latency between different components within SDHI. These performance implications need to be measured against those present in a non-SDHI system, such as virtualization technologies (e.g., hypervisors or containers), to understand if the benefits of a specific SDHI solution overcome those of a non-SDHI solution.

B. SDHI Layer

This layer provides the equivalent functionality of a pure SDI and IaaS in today's cloud. It consists of logical infrastructures that are composed of the assigned HW resources. The operations domain for this layer receives requests for logical infrastructures and composes them by making appropriate calls to the HW-layer functions. This layer has a global view of the underlying HW pools and the composed entities, i.e., it has complete knowledge of all HW resources, how they are connected via the interconnects, how the various resources can be sliced and composed with each other to form logical infrastructures, the logical structures that have already been created out of these resources, and so on. However, unlike a traditional IaaS, this layer does not have any view of the software running on top of the composed HW (unlike OpenStack [72] or Amazon EC2 [79] where the Virtual Machine (VM) provisioning process also includes preparation of a virtual disk from whence the VM starts up). Instead, this layer exposes mechanisms, such as network booting and/or disk authoring, that allow the logical infrastructure to boot into a usable state. Furthermore, this layer exposes smart HW functions (e.g., highly available HW, state synchronization, and live state migration) to the software running on the composed infrastructure, whenever these functions are available in the underlying HW.

C. Platform Layer

This layer handles the software that runs on the composed logical infrastructure to run the application. Examples of open source software platforms include OpenStack, Cloud Foundry [80], Kubernates [74], or a bare Linux OS. In addition to these, there are proprietary platforms, such as [81, 82]. Depending on the platform software, the application may be provided in the form of source code, an executable binary, or even a VM image. A key requirement of this layer is the ability to run multiple platforms at the same time on the same HW infrastructure. This platform layer can be recursive to multiple levels. For example, it should be possible to first deploy an Apache Mesos [83] platform onto a logical infrastructure and then to deploy Apache Yarn [84] and Kubernates on this Mesos platform. This, for example, would allow a MapReduce and containerized applications to execute side by side on the same logical infrastructure. Platform-specific functions, such as auto-scaling of the platform itself, or software libraries that enable the full exploitation of the underlying (disaggregated) HW, are implemented at this level.

D. Application Layer

The application layer handles the specific applications that provide service to the end-user of the cloud. An application can be a VNF [22, 23] that runs in an OpenStack VM or a 4Quant [85] image analysis tool running on Apache Spark. For each deployed platform we assume an application management interface (e.g., Nova-API in OpenStack or Spark driver in Apache Spark) allows deployment and control of applications for that specific platform. These applications and their management functions come from existing software platforms; therefore, it should be possible to run them unmodified on the logical infrastructure. However, extending these platforms with functionalities stemming from HW disaggregation (see section III) may be beneficial.

V. Technologies, Challenges and future directions in SDHI

The direct technological impact of SDHI is confined to the HW and its operation. However, this impact ripples through all the other layers of the architecture. Standardization efforts in the DC technology space expanded with the creation of the open source initiative Open Compute Project (OCP) [41] in 2011. OCP is supported by several industry leaders (IT vendors, network operators, and service providers) and its purpose is to share designs of efficient DC products. Within this initiative, industries have been carrying out extensive collaborative efforts to create new open and flexible standard solutions for future DCs. Among the members of OCP is Intel, who announced in 2013 its hardware disaggregation vision called Rack Scale Architecture (RSA), now called Rack Scale Design (RSD) [40]. RSD is a reference specification for hardware disaggregation and a key component of OCP. RSD's core management and control components (Pooled System Management Engine (PSME), Pod Manager (PODM), and Rack Management Module (RMM)) have been released as open-source software [86]. In early 2016, the Telecom

Infra Project (TIP)) [87] was announced. TIP can be seen as a complement to OCP but is narrowly focused on the development of new telecommunications networking HW. This initiative aims to leverage several OCP efforts, among which are HW disaggregation and composability concepts.

In this section, we present a technology overview and a range of ongoing efforts toward realizing SDHI. These are organized in relation to the different architecture layers presented in section IV. Table IV provides an overview of some of the most relevant efforts. Furthermore, in each section, we present more in-depth efforts and discuss foreseen challenges and future research directions.

A. HW Layer: resource disaggregation and shift in hardware

HW resource disaggregation is in its early stages. Although storage has been disaggregated [15, 41], challenges remain in how to decouple the remaining server components, such as memory, processors, and NICs. This subsection focuses on the inherent challenges, focusing on three main areas: HW component disaggregation (processor, memory, storage, and NIC), resource control (i.e., slicing, aggregation, and sharing), and the infrastructure interconnection fabric.

1) Processor Aggregation and Slicing: The core of SDHI is independent resource pools that can be dynamically composed into logical infrastructures. Whether it is possible to construct logical servers with an arbitrary number of CPUs *without* suffering the overheads of hypervisor-based solutions is one of the key questions [151**?**].

Aggregating CPUs requires a lot of low latency bandwidth for the inter-CPU traffic. Multiprocessor systems based on Intel Xeon processors use QPI [117] for communication between processors. QPI consists of two unidirectional links with a bandwidth of 8-12.8GB/sec and latency of 100-300ns. The upcoming version QPI, called Ultra Path Interconnect (UPI) is designed to run at higher speeds. The big difference of UPI from QPI is that there can be 2-3 UPI per CPU [159]. However, it is hard to realize this bandwidth with low latency over long distances. The common use-case for aggregating CPUs is to build high-capacity servers that are suitable for scale-up scenarios (see Section III-B). However, as current cloud applications are designed to follow the scale-out paradigm, there seems to be little research in this area.

The inverse of aggregating CPUs is slicing a CPU into multiple resources. When the configuration of a server hardware platform is fixed, there is little benefit in slicing the CPU to run different OSs without a hypervisor, as contention exists when accessing hardware resources. However, with resource pooling opportunities exist to exploit CPU slices. Some aspects of this are covered in Section V-D2.

While CPUs are certainly a fundamental element, these are not the only type of processors that are relevant for disaggregation. Acceleration devices, such as GPU and Field Programmable Gate Array (FPGA), are also at the forefront of this technology shift. However, most of the explored approaches purely rely on software and are implemented at the OS level [147–150]. Moreover, Amazon has a commercial offering for remote acceleration by using OpenGL over Elastic Network Interface (ENI).

2) Memory Disaggregation and Pooling: The demand for server memory capacity has been increasing along with the increase in the number of cores per socket, the number of VMs running on a server, and the increased memory footprint per VM [88]. The need for increased memory capacities has been further fuelled by the growing number of in-memory applications. These demands are likely to be further exacerbated by the need for in-memory processing and real-time analytics required by the 20-100 billion [160, 161] connected devices. However, a set of studies (e.g., [162, 163]) reveal that the average memory capacity utilization is 50% and the peak memory usage continues to increase. This combination suggests that every large aggregate amounts of memory will be under utilized.

Moreover, Dynamic Random-Access Memory (DRAM) process scaling is becoming costly, imposing limits on how much DRAM can be placed on a server blade. With pin count on processor sockets not increasing dramatically and the number of Dual Inline Memory Modules (DIMMs) per channel decreasing due to increased channel speeds, the growth in the total addressable memory is constrained.

One method that was used earlier to increase memory capacity was Fully Buffered DIMM (FB-DIMM) [89]. FB-DIMM utilizes the high-speed serial interface between the memory controller and an Ambient Memory Buffer (AMB) residing on the DIMMs and uses the store&forward methodology to increase memory capacity. However, increasing the number of DIMMs results in increased latency. Additionally, extending the serial links beyond a single board requires additional store&forward mechanisms which further increases latency. Furthermore, DRAM placed in a compute blade consumes power whether or not it is used, and the FB-DIMM store&forward electronics further increases heat generation. An extension to FB-DIMM has been proposed by the research community, called Optically Connected DIMM (OCDIMM) [90]. OCDIMM uses optical channels to communicate between the AMB and memory controller to increase memory capacity and reduce latency [90]. Simulation results show that OCDIMM can provide higher bandwidth and capacity. However, OCDIMM requires optical connectivity from the CPU, which is not supported in commercially available CPUs.

At the same time, there is a variety of storage class memories (SCM) [164, 165] that are being researched. These aim to provide diverse capabilities which promise to be beneficial for different workloads. For example, Non-Volatile Memory (NVM) technologies (e.g., [91, 166, 167]) provide increased memory density at a latency higher than DRAM (see Fig. 7).

However, these different memories are only accessible by a single CPU blade, thus complicating the orchestration of workloads and increasing the potential for underutilization of these memories.

Memory disaggregation could help solve these problems

TABLE IV
TECHNOLOGY OVERVIEW OF SDHI.

Type	Area	References	Origin	Scope
Compute Hardware	Memory	[88]	Research	Showing performance benefits of memory disaggregation in memory constrained environments.
		[89, 90]	Research	Increase memory capacity and reduce latency in memory blade servers.
		[91]	Industry	NVM (e.g., 3D XPoint) technologies to increase memory density with a latency higher than DRAM.
		[25, 88, 92–94]	Research	Feasibility of entirely disaggregated memory.
		[95]	Research	Fast in-memory Key-value store.
		[96]	Research	Cost of memory disaggregation.
	Storage	[97–100]	Industry	SAS, NAS, and SAN.
		[101]	Research	Rack-scale storage fabric.
		[102]	Research	Flash Storage disaggregation.
		[103]	Research	Resiliency aspects of using NVM.
	NIC	[104, 105]	Industry	Ethernet multi-host technology for rackscale networking.
		[106]	Research	Software architecture for low latency DC communication.
		[107]	Industry	Integrated NIC Architecture.
		[108]	Industry	SR-IOV network virtualization.
Networking	Interconnection Fabric	[109]	Industry/Research	Roadmap on silicon photonics.
		[110]	Research	Manycore processor-to-DRAM.
		[111]	Research	Switching optically-connected memories.
		[112]	Research	Optical interconnects for disaggregated resources.
		[113, 114]	Industry/Research	Microsoft rack-scale networking.
		[115]	Research	A reconfigurable Rack Network based on SoC switch.
		[116]	Research	Slim Fly a low diameter network topology.
	Interconnection Protocols	[117]	Industry	QPI.
		[118]	Industry	Gen-Z.
		[119]	Industry	OpenCAPI.
		[120]	Industry	CCIX.
	Network Management	[121–129]	Industry/Research	SDN.
		[130]	Industry/Research	SDON.
		[131, 132]	Research	Control Plane OS.
Infrastructure Management	Discovery & Control	[75]	Industry	IPMI.
		[78, 133]	Industry	Redfish.
	Scheduling	[134–136]	Research	Scheduling aspects.
Platform	Cloud Management	[137, 138]	Industry	OpenStack Valence and Ironic/Redfish.
	Network	[17]	Research	Survey on network virtualization hypervisors for SDN.
		[139]	Industry	Cost of network virtualization.
	Virtualization Server Virtualization	[16]	Research	Survey on server virtualization.
		[140]	Industry	QEMU.
		[141]	Research	KSM (Kernel Same page Merge).
		[142]	Research	Minimal hypervisor by efficient OS implementation.
	Operating System	[143]	Research	Hotplug of CPU.
		[144]	Research	Hotplug of Memory.
		[145]	Research	The operating system is the control plane.
		[146]	Industry	NVMe interface to provide access to a storage namespace.
		[147–150]	Research/Industry	Virtualization and Remote Accelerators (DS-CUDA, gVirtuS, GViM, and rCUDA).
		[151, 152]	Industry	ScaleMP and TidalScale.
Research Tools	System Simulator	[153]	Industry	Simics.
	Hardware Simulator	[154]	Industry	CoFluent.
	Data Center Simulator	[155]	Research	Diablo.
	Data Center Simulator	[156]	Research	Firesim.
	Network Emulator	[113]	Industry/Research	Maze.
Projects	Cross-Area	[41]	Industry	OCP
		[87]	Industry	TIP.
		[40, 86]	Industry	RSD.
		[156, 157]	Research	Firebox and dRedBox.
		[158]	Research	M2DC.

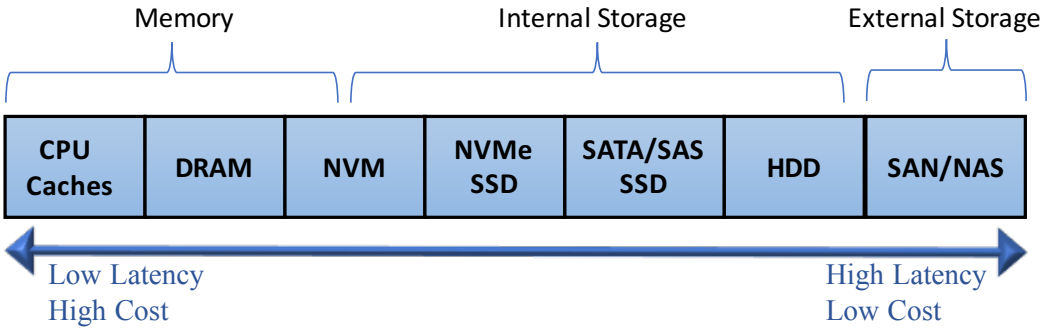

Fig. 7. Memory and storage access latency and cost spectrum.

while increasing the memory capacity available for applications, increasing the average utilization of the infrastructure, and reducing total energy consumption (e.g., [168, 169]). However, memory in current commodity architectures has been very tightly coupled to the CPU due to the very high bandwidth and low latency requirements of the CPU-memory bus (see Table III) and the restrictions on the memory types supported by a given CPU. Moreover, the current memory bus cannot be extended to longer distances due to signal integrity issues. Alternatively, converting the parallel-electrical-bus of the memory devices into a serial-optical-bus becomes a very expensive proposition in terms of both cost and energy.

There is little research that considers completely removing memory from the compute sled. Rather, current research approaches attempt to realize partial memory disaggregation by using local DRAM as a cache for remote memory and swapping pages from/to remote memory as required. In these cases, the interface between CPU and memory is usually based on cache line fetches. Moreover, much of the literature focuses on using page sized memory requests similar to disk-based swapping. This is done to exploit data locality and amortize the latency incurred due to delays over the interconnect. Two methods are widely used in the literature: (1) including a PCIe extension to bridge multiple memory domains or (2) use of Remote Direct Memory Access (RDMA) (e.g., Infiniband [170] or Omnipath [171]) or other low latency methods. These approaches have been shown to work well for applications with good data locality [92]. Quite a few papers have evaluated these approaches, some of these evaluations are described in the following paragraphs. Table V summarizes the efforts toward memory disaggregation.

K. Lim, et al. [88] designed a new general-purpose architectural building block - a memory blade - that allows memory to be disaggregated across a system ensemble. This solution can be used for memory capacity expansion to improve performance and for sharing memory across servers. They claim their solution can potentially reduce the cost of provisioning and reduce power consumption. They proposed two new system architecture solutions: (1) a page-swapped remote memory at the virtualization layer and (2) a cache level-access remote memory with support for transparent memory expansion and sharing for commodity-based systems. Moreover, they explored the implications of these solutions by developing a software-based emulation platform using the Xen hypervisor [93] and compared their results with a disk-based swap mechanism.

Silicon photonics [109] is an evolving technology in which data is transferred optically. This technology is proposed as an interconnect choice for rack-scale systems. Several papers (e.g., [110, 111]) suggest using silicon photonics for CPU-memory interconnects. However, commercial 100 Gbps silicon photonics are insufficient for CPU-memory interfaces, as 1 Tbps connections may be required to fully disaggregate memory. Assuming four memory controllers per CPU (i.e., the norm in X64 Xeon CPUs deployed in clouds), one would need at least 250 Gbps photonic interconnects to meet current requirements. However, this level of performance is expected to be commercially available in a few years when these interfaces are directly integrated into CPUs. However, yet more time will be required for these solutions to be cost effective for broad adoption across the entire infrastructure.

P. X. Gao, et al. [25] demonstrated that current networks are sufficient to do memory disaggregation for a variety of applications. They claim that silicon optics and other future-looking technologies are not required and that current technologies are sufficient to realize memory pooling. One of the main assumptions is that NICs will be embedded into CPUs to reduce latency in a disaggregated memory environment. Their simulation study assumes the use of 40-100Gbps Ethernet connections. Moreover, they ignore software overhead associated with page operations despite the fact that this overhead could be non-negligible if a running process has to do a context switch while waiting for the page swap operation to complete.

Marlin [94] is a PCIe based system where all the memory from multiple blades is aggregated into a global memory. Each machine in the rack is connected to a port of the PCIe switch through a PCIe expansion card and a PCIe cable. On each Non-Transparent Bridged (NTB) port of the Marlin switch is a DMA engine capable of initiating DMA transactions across different physical address ranges. Marlin refers to this Hardware based RDMA (HRDMA) as providing a cross-machine memory copy operation. HRDMA allows data to be copied from one application process address space (running on one machine) to the address space of another

TABLE V
SUMMARY OF INDIVIDUAL PAPERS ADDRESSING MEMORY DISAGGREGATION.

Ref	Objective of work	Evaluation method	Result Summary	Interconnection	Comment
[88]	Evaluate feasibility of memory disaggregation.	Simulations & Prototype	Memory disaggregation is feasible and can provide substantial performance benefits (on average 10X) in memory constrained environments.	PCIe	OS & hypervisor transparent but requires custom chip.
[25]	Evaluate the feasibility of memory disaggregation with existing system designs and commodity networking technology.	Simulation & Emulation	Current network technologies are sufficient to do memory disaggregation for a variety of applications but with some performance degradation.	RDMA	-
[94]	Present the design, implementation, and evaluation of a PCIe-based rack area network system for memory disaggregation.	Prototype	Based on their design, one-way kernel-to-kernel latency is 8.5 μsec, and the end-to-end sustainable TCP throughput is 19.6 Gbps.	PCIe & HRDMA	-
[93]	Explore software and systems implications of disaggregated memory.	Prototype & Emulation	They develop a software-based prototype by extending the Xen hypervisor to emulate a disaggregated memory design wherein remote pages are swapped into local memory on-demand.	NA	-
[92]	Feasibility of fully disaggregated memory.	Prototype	Memory disaggregation is possible under Spark SQL workload with already available commercial network technology.	PCIe	A case study on Spark SQL.
[110]	Present a new monolithic silicon photonics technology and its application for manycore processor-to-DRAM networks.	Simulation & Prototype	The focus was on network aspect for communication between CPU and memory based on photonics.	NA	-
[111]	Assess feasibility of transferring data across processors by using the optical interconnection fabric.	Simulation	The performance data demonstrates the effectiveness of switching memory in transparent data sharing and communication within a rack.	NA	-
[172]	Design and implementation of a new memory distributed computing platform.	Prototype	The platform performs well and shows order of magnitude better throughput and latency than main memory systems that use TCP/IP on the same physical network.	RDMA over RoCE	-
[173]	Memory architecture for clusters to enable a distributed non-coherent shared-memory view of the memory resources present in the cluster.	Platform	The database that is running in this prototype beats commercial solutions in terms of latency and throughput.	HyperTransport [174]	FPGA based solution
[175]	Design a system using commodity products that connect multiple nodes and enable resource sharing among these nodes.	Prototype	The evaluation results indicate that resources can be efficiently shared in many cases.	PCIe SR-IOV	Software stack extension.
[176]	Design and implementation of a remote memory paging system for an RDMA network.	Prototype.	Demonstrate the overall memory utilization increases in a cluster of nodes.	RDMA	-
[177]	Assess how can Symmetric Multi-Processing high memory demands take advantage of remote memory.	Prototype	The proposed solution significantly improve the performance of memory intensive workloads	RDMA	-

application process that is running on another machine in the same rack without any software intervention.

There are software solutions that aggregate memory across multiple physical servers and provide access to an aggregated memory, such as [152] and [151]. These solutions target big data applications. However, performance penalties are high when cross-server communication occurs. Moreover, these solutions require adding server boards containing both CPU and memory to increase either of these resources (as the software has to have a CPU to run on to access the physically local memory to make it available remotely).

The design of a memory blade has inherent challenges. As applications become more resource demanding and require larger working sets, several memory pools will be needed on one memory blade to address the increase in required memory bandwidth and to reduce noisy neighbor problems. The rate at which applications need memory, the speed of interconnects, the memory node characteristics, and the performance of memory controllers are some of the factors determining memory pool capacities and the number of required memory controllers. The location of a memory controller itself presents interesting choices. A memory controller placed close to the memory blade can perform memory bank refreshes locally, thus optimizing the power utilization and logic on the CPU socket. Conversely, the absence of a memory controller on the CPU could present other challenges due to the absence of an on-chip memory arbitration mechanism. A future design could potentially split the roles of the memory controller and realize different roles at different locations.

With memory disaggregation and pooling, different compute nodes and devices can access and share memory blades, thus memory access control mechanisms will be required to prevent unauthorized bare metal servers or devices from accessing memory to which they should not have access. If

a hypervisor exists on the compute nodes, it could provide the needed protection. However, in bare metal environments, such access control is required either at the hardware level of the compute node or at the memory blade. Moreover, as multiple applications could attach to the same memory pool, quality of service considerations are required at various points. Additionally, protocols between the compute blade and memory blade need to be agnostic to current memory protocols (e.g., DDR3 and DDR4) to allow the use of different types of memory in the memory blade.

Disaggregation brings the advantages of modularity, independent component replacement, and right scaling, to the memory system. Additionally, it can enable: (1) redundant memory copies based on application requirements rather than being limited to what is supported by a particular CPU; (2) reduction in the number of refresh requests issued by the memory controller that is part of the CPU to a memory controller placed on the memory blade, thus potentially increasing the scalability and energy efficiency as memory capacities increase; (3) runtime change of memory banks on a memory sled to provide better performance or to reduce energy consumption; (4) merging of similar pages across different compute sled boundaries; and (5) shared memory architectures for intra-server communications. For example, this disaggregation could enable efficient VM migration (e.g., [178]) within the same rack or sharing of system state across several logical services running within the rack without employing load balancers.

Lastly, there are a few other important aspects to keep in mind. The compute nodes and the memory sleds have to be placed reasonably closely since each meter of separation between two resources adds at least 4-5 ns of latency, thus the separation between memory and CPU may not exceed a few meters for full memory disaggregation. Moreover, memory disaggregation places new requirements on OSs and applications as to what memory disaggregation means when compared to a traditional Non-uniform memory access (NUMA) system (see Section V-D1). For example, application orchestration in an SDHI needs to pick components such that resources are localized at an optimal distance from other resources (e.g., CPU, memory, NICs, and GPUs). Furthermore, the OS/application software needs to be informed and be able to understand the various latencies to achieve optimal performance. In a VM environment, the guest kernel might need to know about the existence of local and remote memories so that the appropriate memories can be assigned (e.g., a kernel driver might need local memory). This will have implications on the huge translation lookaside buffer (TLB) [179] setups used by big data applications to improve performance, as they need consecutive pages to be loaded (or possibly flushed) at the same time. While some disaggregated memory access schemes allow cache line access, the application could incur higher latencies. In these approaches, the CPU needs to do polling for I/O operations to fetch remote memory because I/O completion times are expected to be few microseconds (i.e., typically less than the time slice that processes are assigned). Having to perform a CPU context switch rather than being able to poll could have an adverse performance impact. In turn, this could affect the use of hardware provided hyperthreads which are very helpful in increasing the effective throughput of the CPU while hiding the latencies of transfers between data memory and cache.

3) Storage Disaggregation and Pooling: Direct-attached Storage (DAS) gives the best performance in most cases due to the absence of any network protocols and the need for reliably communicated acknowledgments from the remote end point. However, DAS has several problems, e.g., each directly attached storage device needs its backup, spares, and time from management personnel or automated management. Another disadvantage is that access to the stored data depends upon the availability of the compute node that it is attached to, hence, if the attached compute node is unavailable for any reason the data will be inaccessible. For these reasons, the storage system was disaggregated early [97] to avoid fate sharing of data with computing, and there are ongoing efforts to improve the performance and scalability of Storage Area Networks (SANs) and Network Attached Storage (NAS) [98–100] (see Fig. 8).

Initially, storage disaggregation was implemented using the network; for example, via Fiber Channel (FC) SANs. FC used a system of negotiated end-to-end credits and data was assured safe passage through the network. With the wider adoption of DC technologies, running two parallel networks, one using FC and another using Ethernet, has become a problem. Different protocol extension have been proposed to overcome this issue. For example, both FC and the Small Computer System Interface (SCSI) protocol have been extended to run on top of Internet Protocol (IP), leading to Fiber Channel over IP and Internet SCSI (iSCSI) so one network can be used for both storage and other DC traffic. Additionally, extensions have been proposed to Ethernet to introduce DC Bridging (DCB) to provide guarantees similar to FC. The Storage Networking Industry Association [180] is working to expand the adoption of these technologies.

Commercial solutions were developed for hyper-converged enterprise clouds that packaged compute and storage resources together to gain the benefit of direct attached storage. However, these solutions suffer from the problem of matching just the right amount of computing and storage. As data is added, storage typically runs out, and then when new nodes combining computing and storage are added this leads to an imbalance between the compute and storage resources with respect to the actual requirements.

While hard drives have a latency of few milliseconds, newer solid-state drives (SSDs) have a latency of a few microseconds. The current generation of PCIe SSDs offer higher performance and lower latency. When SSDs are accessed across the network, their high performance is reduced due to latencies of the network and protocol stack. PCIe or Serial Attached SCSI can be extended over an optical bus to make it appear to be direct-attached storage with native performance. In this manner, the amount of computing and storage can be changed over time to create better hyper-converged solutions. In current object storage solutions, the storage client and storage server talk over the network

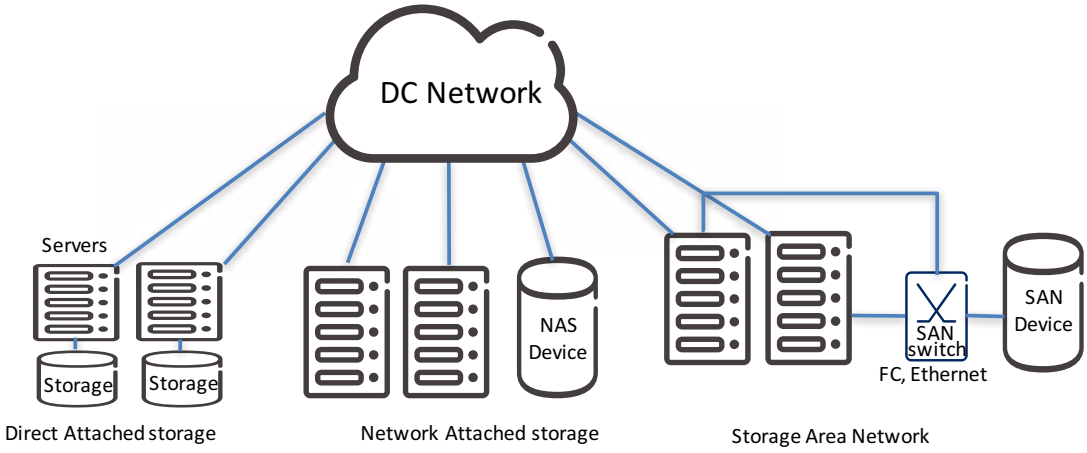

Fig. 8. DAS, SAN, and NAS architectures.

while the disk can be either direct-attached or connected over the network. Disaggregated systems potentially allow for all of these elements to be connected without a network stack adding to the latencies. D. Ruiquan et al. [101] show a rack-scale storage fabric displaying the benefits of disaggregation along with the ability to create the required compute and storage ratios with performance equivalent to DAS, but without its problems. Flash Storage Disaggregation [102] disaggregates flash storage over an Ethernet network and shows that disaggregation allows CPU and flash storage resources to scale independently in a cost-effective manner. A. Klimovic, et al. attribute a 20% drop in application performance due to disaggregation [102]. They attribute this to application software overheads and to the throughput and latency of the iSCSI protocol.

As mentioned previously, new SCM memories that can provide DRAM-like latencies and persistent storage with much higher storage density are expected to be available soon. When available, they will provide cache line access to data as opposed to block-level data access that traditional storage devices offer. However, their effectiveness will be reduced if they are accessed via the network. At the same time their utilization is likely to suffer if they are placed on a compute blade, while rack scale disaggregation could provide better utilization. Another issue relates to the fact that most DCs would like to keep a copy of the data outside of the rack for resiliency purposes, but this can add significant overhead and remains a subject of research [103].

4) NIC disaggregation: In earlier DCs, Infiniband was a common choice for SAN, while Ethernet was the choice for communication networks. Over time Ethernet has evolved to support higher transmission rates - first with 1Gbps, subsequently 10Gbps, 40Gbps, and today there are efforts to achieve 100Gbps. While Infiniband is still widely used in specialized environments such as HPC, Ethernet's evolution makes this technology a compelling alternative to Infiniband and suitable for both DC wide network and SAN [181].

NICs play a significant role in server disaggregation as the physical limitations of other interconnects (e.g., the number of PCIe lanes and the distance they can reach) impose limits on which resources are directly connected and which resources can be accessed via the network. New NICs support various protocols and operations, for example, crypto offloading, RDMA, OpenFlow [182], and P4 [183], thus reducing the load on the CPU. Infiniband [170] is a technology of choice for environments that require RDMA and sharing a memory from remote hardware components without the intervention of the local CPU. On the other hand, RDMA over Converged Ethernet (RoCE) and internet Wide Area RDMA Protocol (iWARP) are two RDMA protocols supported by vendors on top of Ethernet. Although Infiniband still has better latency numbers, DCs are reluctant to run two separate network technologies, thus predominantly deploying Ethernet-based RDMA. However, there could be a case to utilize Infiniband as an infrastructure component inside a rack to facilitate transfers with accelerators and remote memory without applications being aware of it. This reduces the DC administrator's pain to deploy two different network technologies while intra-rack communication can benefit from Infiniband.

NIC disaggregation can be useful in composing servers with the requisite amount of network bandwidth. NIC disaggregation makes it easy to exploit other benefits of disaggregation, such as independent lifecycle management, accelerating specific applications, and reducing the inventory of specialized cards. Currently available disaggregated NICs, commercially called multi-host NICs [104, 105], simultaneously support multiple physical servers [184] and allocate variable amounts of bandwidth to the different hosts. These multi-host NICs also reduced the number of ports needed on the Top of Rack (ToR) switches. Moreover, with OpenFlow support, intra-VM traffic between different hosts attached to the same multi-host NIC can bypass the ToR switches.

The above considerations may necessitate the use of integrated NICs for latency critical traffic together with disaggregated NICs for non-latency critical traffic to/from the same server.

B. HW Layer: Networking

Networking plays a pivotal role for a resource disaggregated DC and is seen as an enabler as well as an inhibiting factor. The network of a resource disaggregated DC has to support, in addition to all of the network traffic corresponding to today's DC, the low latency and high bandwidth requirements for communication among components of logical servers. Two distinct communication paradigms with different requirements exist in a disaggregated architecture: (i) communication external to the logical host (i.e., via a DC-wide network) and (ii) communication internal to the logical host (i.e., via an interconnection fabric). The challenges of networking in a resource disaggregated DC can be divided into five parts: challenges for DC-wide networking, challenges for the interconnection fabric, challenges of using the DC-wide network as an interconnection fabric, challenges due to the gap between end host networking, computing, and data access time, and finally challenges due to hardware interconnection protocols. Table VI summarizes some of the most relevant works related to SDHI's networking.

1) DC-wide Networking: Similar to today's DCs, inter-logical host communication uses the logical host's NIC(s) and the DC-wide network (with its physical or virtual switches, routers, and other chains of network functions [123]) and utilizes standard networking protocols (e.g., TCP/IP) for communication. Although this communication benefits from dynamic network resource provisioning capability, the dynamicity of this communication is limited by capabilities of network control approaches (e.g., SDN [17]) with respect to how rapidly it can reconfigure the network. This aspect of DC networking and how to improve the SDN has been well studied [121–128].

2) Interconnection fabric: As described earlier, intra-logical host communication uses the DC's interconnection fabric so that each resource (or pool of resources) can communicate with other resources through shared buses. Thus the communication that was previously restricted to a physical server's motherboard is carried across this fabric. Therefore, more bandwidth is needed on shared buses (such as PCIe). The interconnection fabric must ensure sufficient performance that the (logical) server performs at the desired performance level. As the DC's interconnection fabric must provide connectivity between various types of resources, the burden of enabling disaggregation falls on it.

The appropriate interconnection fabric for a resource disaggregated DC should provide low latency, high bandwidth, high resiliency, low cost, and low power consumption. These requirements suggest an optical network is a promising means to overcome the practical limitations of copper board traces and the very strict demands upon latency and bandwidth when interconnecting the various components (as described in Table III). This requires cost-effective and efficient optical network equipment to ensure efficiency in DC based on SDHI. Furthermore, when an optical network is used to interconnect different resources (e.g., CPUs and memory), the capability of these resources to natively work with optical communications is beneficial as it eliminates the cost of optical-to-electrical conversions and the corresponding energy inefficiencies. These resources also expected to originate wide range of wavelengths to support more point to point interconnections (without doing costly intermediate switching and routing)

In addition to the low-latency and high bandwidth requirements provided by the optical overlay, such an interconnection network needs to be highly dynamic and flexible. This means that such an optical overlay requires an efficient network management mechanism that can establish and reroute optical paths quickly. Consequently, separation of control and data plane for a DC's interconnection fabric could be highly beneficial, thus SDN-based optical networking is a potential solution for managing such a network fabric. Combining optical overlay and SDN results in a software-defined optical network (SDON). SDON leverages the flexibility of SDN control to support network applications via an underlying optical network infrastructure. A thorough survey regarding existing SDON architectures and technologies is presented in [130].

In contrast to inter-logical host communication, the intra-logical host communication does *not* require frequent changes, as the configuration of a logical host remains relatively constant (except when optimizing and re-scheduling a logical host's resources). A resource disaggregated DC's fabric is expected to utilize a variety of interconnect technologies and protocols (i.e., QPI, SATA, PCIe, and Ethernet). However, with regards to fiber-optic interconnects between different resources in a resource disaggregated DC, much tighter integration of optics with the resources is required (e.g., optical PCIe and other optical data links and networks) [112]. It should be noted that achieving the appropriate trade-off between reconfigurability & dynamicity of the fabric's topology and the cost, power, and number of required ports of the switching fabric is an important consideration when designing a new architecture, especially for intra-rack connectivity. Moreover, the solution is expected to vary depending on the number of endpoints to be connected. Thus challenges will arise when the number of connection endpoints scales (e.g., to several hundred or thousand).

Some fundamental questions relevant to the interconnection fabric are: "How to meet the requirements for I/O-CPU, CPU-CPU, and CPU-memory communication?"; "What is an appropriate network and system architecture (including physical network layout e.g., two-tier heterogeneous or flat network architecture) for a resource disaggregated DC with suitable abstraction (i.e., packet-switching or circuit-switching, best-effort or reliable communication)?"; "Is there a need to modify/enhance the interconnect protocols to optimize communication?"; and "How to share the interconnection fabric resources between different traffic types requesting communication among various resources within logical servers (e.g., CPU-to-CPU, CPU-to-RAM)?".

It is possible that the interconnection latency requirement cannot be met even with the low-latency and high bandwidth optical network (e.g., 10 m network distance can add 40 ns of latency). Hence, software optimizations that intelligently

TABLE VI
SUMMARY OF INDIVIDUAL PAPERS ADDRESSING NETWORKING FOR RESOURCE DISAGGREGATION.

Ref	Objective of work	Evaluation method	Summary of Result	Comment
[112]	Discuss latency in optical data links for optically attached memory, optical PCIe and other optical data links and networks.	NA	The discussion shows that optics-induced latency will contribute only marginally to the overall latency in communication.	-
[114]	Rack network design and network stack for rack-scale computers (routing and rate control).	Simulation	They proposed a rack architectures. The result is showing how the characteristics of the proposed rack architectures allow for new approaches that are attuned to the underlying hardware.	Same network fabric for both IP and non-IP traffic.
[113]	Proposing a network stack for rack-scale computers that provides flexible and efficient routing and congestion control.	Prototype & Emulation & Simulation	The result shows the proposed solution achieves very low queuing and high throughput for diverse and bursty workloads while routing flexibility can provide significant throughput gains.	Follow up from [114]
[115]	Design a rack-scale network that reconfigures the topology and uplink placement using a circuit-switched physical layer.	Simulations & Prototype on ASIC switch	The proposed solution optimizes the network's physical topology and significantly outperforms static topologies and has a performance similar to fully reconfigurable fabrics.	No ToR Switch.
[111]	Assess the feasibility of transferring data across processors by using the optical interconnection fabric - without physically moving the data across electrical switches.	Simulation	They proposed memory switching protocol that allows large-scale data communication across processors through the transfer of a few tiny blocks of meta-data.	Rack-scale.
[94]	Present a PCIe-based rack area network solution to support the communications and resource sharing needs of disaggregated racks.	Prototype	The proposed solution includes a hybrid ToR switch that consists of PCIe ports and Ethernet port. The solution supports HRDMA as communications primitive between servers within a rack and supports socket-based communications for legacy network applications.	-
[25]	Evaluate the network requirements for resource disaggregation concerning application bandwidth and latency demands, and requirements imposed by resource disaggregation.	Simulation & Emulation & Prototype	Results showed that resource disaggregation is feasible even with existing network hardware and that the application's memory bandwidth demands determine the scale of disaggregation.	Rack-scale & DC-scale

use caches to hide these latencies can be useful. Research in OS or middleware to place and adjust data in the appropriate hierarchy (e.g., based on access patterns) without application involvement would be needed (see Section V-D1).

3) A DC-wide network as an interconnection fabric: An alternative way to realize a resource disaggregated DC's network is to utilize the DC-wide network as an interconnection fabric (rather than having two separate networks). To operate such a DC-wide network requires a suitable network controller. This controller could be based on distributed control software, a logically centralized controller as network OS (i.e., in-line with SDN philosophy [125]), or a combination of both approaches (e.g., see [126]). Moreover, to get the most out of the entire environment, we foresee that networking resources should be handled tightly together with the other types of resources when composing logical instances. As a result, we expect that networking constraints play a fundamental role as part of a DC's resource scheduling (see Section V-C2).

Apart from the interconnection fabric's challenges, some other questions need to be answered, such as: "How to perform routing and congestion control between different entities in different resource pools or even within one pool?" and "How to share network resources between inter-logical servers and intra-logical server traffics?".

Some work has already been carried out on rack-scale networking that targets in-rack consolidation, where the objective is to increase the density of servers inside a rack [113–115]. These works claim that with a dense deployment of microservers in a rack, it is unlikely that you can have a ToR switch or multi-tiered topology (due to the required high number of ports and bandwidth). This leads to the elimination of the ToR switch and direct connection of the microservers in the form of System on Chips (SoCs). Along with this line, P. Costa et al. [113, 114] assumed a single network for both IP and non-IP traffic based on a distributed switch where the microservers are interconnected through their NICs with a multi-hop direct connect topology (as opposed to tree-like topologies of today's DC). In this case, each node functions as a switch forwarding packets. This work mainly addressed two challenges in intra-rack communication: routing and network sharing. They proposed R2C2, a network stack that provides routing and congestion control for very densely connected servers within a rack. Simulating the proposed network stack across a rack, they showed that R2C2 provides low queuing delay, high throughput, and flexible routing among the microservers. However, it is unclear whether this approach is applicable to resource disaggregation where each microserver can access the resources of other microservers.

S. Legtchenko et al. [115] assumed microservers (or SoCs) would be connected through an embedded switch in the System on Chip (SoC) (assuming that the SoC has an embed NIC/packet switch). To eliminate the problem of multi-hop routing in such a topology, they proposed

a partially reconfigurable in-rack network, called X-fabric. X-fabric uses a combination of layer two packet switching (via the embedded packet switching in the SoC) and layer one circuit switching (their home-made switch fabric was based on electrical signal forwarding without any queuing or additional packet inspection). This solution supports the interconnection of approximately 350 microservers within a rack. Packet loss may occur in the system at reconfiguration time (roughly 30 ns, equivalent to losing one packet while reconfiguring), and they rely on transport layer protocols (such as Transmission Control Protocol (TCP)) to address this packet loss. They proposed an algorithm for reconfiguring the circuit switching topology based on a rough estimate of the demand matrix of the microservers. M. Besta and T. Hoefler [116] introduce Slim Fly, a general approach that can produce a new class of topologies that rely upon high-radix routers and cost-effective fiber optics. This solution formulates the topology requirement as a mathematical graph optimization problem and defines how different endpoints within the DC should be connected with the minimum diameter to achieve a low-latency, low-cost, and high-bandwidth network topology. This approach seems to address the challenges of resource disaggregated DC networking.

P. X. Gao, et al. [25] extend the work by S. Han et al. [24] and focus on the network requirements for resource disaggregation. They investigated the feasibility of disaggregation based on application bandwidth and latency demands, together with requirements imposed by resource disaggregation. Their investigation was based on partial CPU-memory disaggregation where each CPU has some local memory that acts as an additional layer of cache, while the disaggregated memory acts as an expanded memory in the memory hierarchy. Further, they assumed each CPU's cache coherence domain is limited to a single compute node, hence there is no direct CPU-to-CPU traffic cross the resource disaggregated DC network. They also assumed VMs as a host abstraction where each resource of this host is disaggregated. They experimented by executing different types of applications (e.g., memory, I/O, and compute intensive) over an emulated resource disaggregated DC with a flat network architecture and commodity HW. Their results showed that resource disaggregation is feasible even with existing network hardware and that the application's memory bandwidth demands determine the scale of disaggregation (i.e., rack-scale or DC-scale). Moreover, their results showed that meeting the latency requirement in a DC's network based on SDHI is more challenging than providing bandwidth and that the central latency bottleneck is the networking *software* rather than hardware.

The level of disaggregation in future DCs will, to a great extent, be determined by the maturity of the networking technology. Assuming the speed of light is the only limiting factor in the communication latency, the latency requirements (within a single logical host) of CPU-to-CPU and CPU-to-memory communication (see Table III) imply that the maximum distance between two CPUs cannot exceed 6 meters, while the distance between CPU and memory can not exceed 12 meters. This ignores other sources of latency, such as switching, serialization, and optical-electrical encoding/decoding. Although the delay and bandwidth depend directly upon the network, one should not neglect the important role played by individual components such as the end host's networking stack and memory access time.

4) Gap between end host networking, computing, and data access time: In the communication between a CPU and a particular memory (whether it be a volatile or non-volatile memory), there is high latency to access the first byte of non-local data, which results in a considerable number of CPU instruction cycles being wasted. Fig. 9 shows those components that contribute to the latency to fetch the data from remote resources.

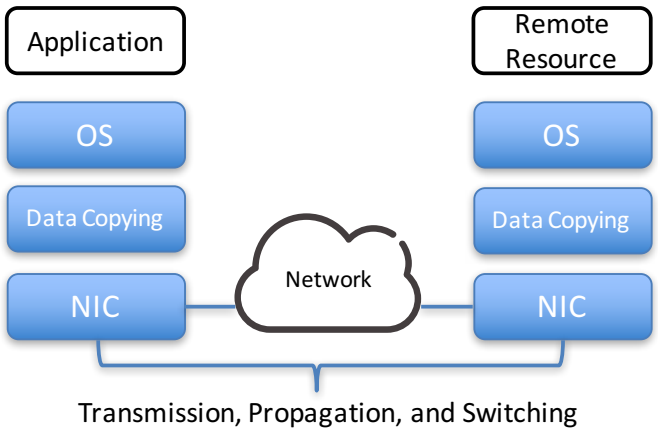

Fig. 9. Key components that contribute to the latency to fetch the data from remote resources.

One alternative is to move part of the processing to the data, rather than moving the data to the processor. For example, data can be pre-processed with simple processors co-located with the memory pools. Additionally, performing a topological sort and pre-fetching the data in a manner that takes into account the memory hierarchy (i.e., by utilizing access pattern prediction intelligence), enables the relevant data to be closer to the CPU (e.g., in Random-Access Memory (RAM) co-located with a CPU or in the CPU's cache), which in turn can eliminate the initial access delay and improve the CPU's data locality, hence reducing the CPU's access time to data (see [185]).

As discussed earlier, networking *software* and end host's network stack will be another source of latency for communication between entities. One can address this at the HW layer by enhancing the current NUMA architecture (e.g., see [49]), or propose a new computer architecture (e.g., see [186]). Alternatively, the problem can be addressed at the software layer. In current OS design, the kernel mediates access to I/O devices, hence it is in the critical data path between the application and hardware, but this adds a bottleneck and increases the end-to-end DC networking latency. To eliminate this bottleneck, one can take advantage of Single-Root I/O Virtualization (SR-IOV) [108] or remove the kernel from the data path by placing some kernel

functionality in the application and some functionality in the I/O devices (see [131, 132]). Current I/O devices are powerful enough to perform some actions to reduce end-to-end latency. For example, many NICs can perform pre-processing of packets (e.g., packet filters, rate-limiting, and routing the packet to a dedicated CPU), which when exploited reduces end-to-end latency in a DC-wide network (e.g., see [95, 187, 188]).

5) Hardware Interconnection Protocols: With the decoupling of HW components at the very core of this disruptive evolution, it is essential for the hardware interconnection protocols to enable this disaggregated architecture. In this sense, initiatives such as Gen-Z [118], Open Coherent Accelerator Processor Interface (OpenCAPI) [119], and Cache Coherent Interconnect for Accelerators (CCIX) [120] have been created. These initiatives promise to ease the transition to more flexible, scalable, and high performance DC infrastructures.

Today, the communication between devices in a server system is done through specialized/dedicated buses, for example, a processor or SoC has embedded media controllers that use specialized media buses to communicate with the respective media. Gen-Z is a new and open data access technology being proposed by an industry-led consortium. It is a semantic memory fabric that can be used to communicate with every device in a server system. With Gen-Z, media-specific functions are decoupled from external devices and placed along with the associated media devices. Fig. 10 shows in the case of SoC and memory with the memory controller is placed next to the memory devices. This simplifies device communication by making it media agnostic, thus allowing devices to become effectively decoupled/disaggregated. Ultimately, all devices can become peers to one another and speak the same language - see Fig. 11. Moreover, Gen-Z claims to offer both low latency and high bandwidth thus allowing computing systems to match the growing capabilities of low latency devices, such as SCM [164, 165].

As applications continue to demand better performance when moving data across various HW components, acceleration engines have started to play an increasingly important role, because of both their performance benefits and positive impact on reducing DC power cost and space. In this sense, OpenCAPI and CCIX are working to improve performance and simplify the communication between HW devices. OpenCAPI is an open interface that is agnostic to processor architecture, hence it allows any microprocessor to attach to coherent user-level accelerators, I/O devices, and advanced memories accessible via read/write or user-level Direct Memory Access (DMA) semantics. Similarly, CCIX allows multiple processor architectures and accelerators to seamlessly share data, allowing two or more devices to share data in a cache coherent manner. The standard allows processors based on different instruction set architectures to extend their cache coherency to accelerators, interconnects, and I/O.

The three consortiums (Gen-Z, OpenCAPI, CCIX) share common goals and aim to increase the interconnect bandwidth between HW devices while reducing latency and enabling data coherency between devices. While Gen-Z primarily focuses on a routable interconnect that could be applied at the rack or DC level, OpenCAPI and CCIX have a more confined and overlapping space of interest. It is, therefore, reasonable to

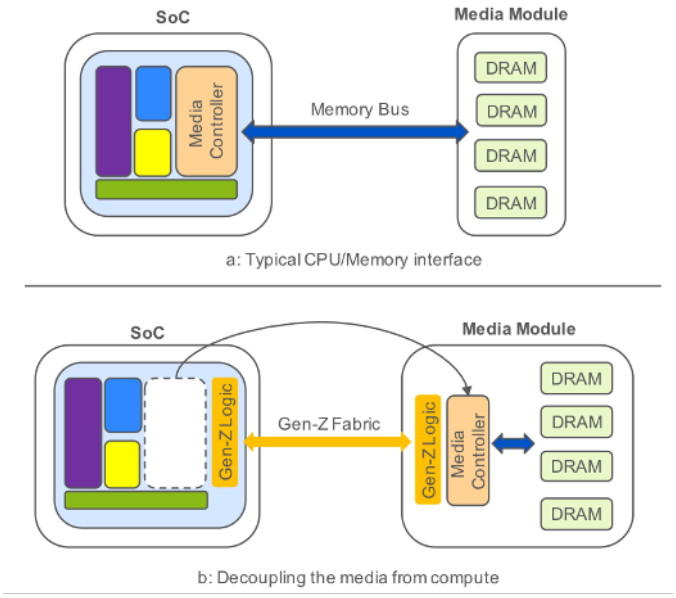

Fig. 10. Existing SoC with DRAM and an SoC with a Gen-Z "logic" Media Controller. Figure adapted from [189].

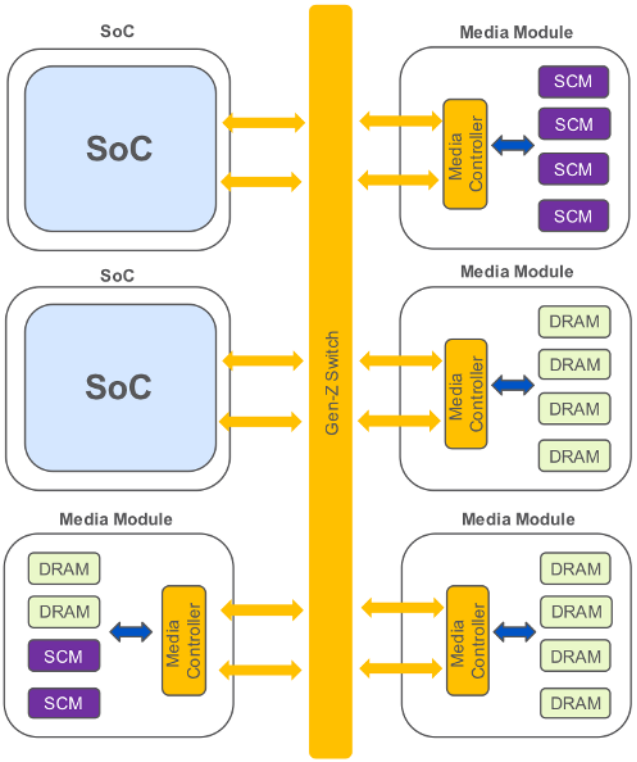

Fig. 11. Gen-Z architecture illustrating a diversity of media devices. Figure adapted from [189].

expect that over time some convergence will occur, mainly between OpenCAPI and CCIX.

C. SDHI Layer

The basic principle of SDI, where an infrastructure is defined by software, is not entirely new *per se*. This concept has been applied and explored in different ways. Virtualization technologies besides SDN and SDS have been a way to realize SDI (see Section II), not in its pure sense (i.e., at the HW level) but at a software level that allows emulation, to a certain extent, of SDI. Only now, with the concept of HW disaggregation, SDI is explored in a purer sense as SDHI, as we refer to it in this article. In the case of a resource disaggregated DC, SDHI aims to enable dynamically define and re-defining of logical server systems. This subsection elaborates on the vision of SDHI in general and its corresponding challenges.

1) Infrastructure Discovery and Control: Among the most fundamental requirements in a disaggregated environment is that the SDHI management layer must be able to dynamically discover, monitor, and control the infrastructure. Within the current SDI environment, traditional servers are limited to thos e resources integrated on their physical motherboard (or directly connected to this motherboard). Physical motherboards aggregate and expose capabilities and monitoring information about the HW associated with a server. Additionally, they also expose an extremely limited set of control features (e.g., turn on/off system and reboot system). Today these operations are typically available via the widely adopted Intelligent Platform Management Interface (IPMI) [75].

Management complexity increases in an SDHI as resources are distributed and logically associated to form logical servers. The type of information discovered, exposed, and monitored by the SDHI layer needs to include (but is not limited to): hardware characteristics, capabilities and monitoring information; connectivity map of the hardware resources; information to (directly or indirectly) derive compatibility between different hardware resources when composing logical infrastructures out of the available resources (e.g., to know that processor 1 in blade A can interact with the memory block 1 in blade B, but not with memory block 2 in blade C); existing composed logical infrastructures and their associated monitoring information; and HW resources available for composition into existing or new logical infrastructures. This means that new types of information needs to be collected and much more information needs to be exposed and handled. Moreover, these control features must include physical hardware component (e.g., processor, memory, networking) partitioning/slicing and interconnection of physical hardware component partitions (including configuration of the DC's interconnection fabric).

At a local hardware level, these functions are very simple, allowing them to be implemented in microcontrollers, e.g., Board Management Controllers. This local control implements only the essential functions, relegating more sophisticated features (including those that require non-local knowledge) to higher level SDHI management entities (e.g., a rack or cross-rack management entity). Overall, these cross-level functions must be extremely scalable, allowing them to efficiently handle large-scale systems. This means that no single entity has a complete real-time detailed view of the entire infrastructure.

To a certain extent, some of the challenges relate to the work being carried out by the Distributed Management Task Force (DMTF) Scalable Platforms Management Forum (SPMF) Working Group [133]. DMTF is working on the specification of Redfish [78], an open industry standard specification and schema that aims to enable simple, modern, and secure management of a scalable HW platform. Redfish is seen as the long-term direction for node and sub-node level management, replacing today′s widely used IPMI. The latest release of their specification introduced initial support for fully disaggregated environments by exposing individual resources (e.g., processor, memory, etc.) & resource pools, rather than only traditional servers; and a composition service that allows composition of logical servers by explicitly stating which physical resources are to be used. However, there remain a set of unsolved issues.

Some of these issues are: "How to autonomously determine compatibility between different hardware devices?" As this has not been considered in Redfish, because Redfish assumes there is an external process through which HW compatibility is explicitly provided to the system and then simply exposed by Redfish. We expect that from a composability perspective, Redfish will evolve to support logical server composition *without* the need to explicitly state the exact physical hardware components to be used, as is currently required. This will allow the SDHI layer to optimize device allocation transparently to the higher layers.

Another important aspect is DC networking and fabric device exposure/control. Current work within Redfish on this aspect is at an early stage, but indicates that control of network devices will be done by mapping YANG models to Redfish models [190]. The intention is that network devices will be managed via the Redfish RESTful interface, regardless of the YANG model they support.

While Redfish promises to be a strong foundation for future infrastructure discovery and control mechanisms by providing a set of standard interfaces that can be applied at different levels of the infrastructure (e.g., resource level, chassis level, rack level, and DC level), it is important to keep in mind the open question: How to use and best propagate the information throughout the system?.

2) Scheduling and Placement: Scheduling of workloads in a DC has received some attention in the literature in recent years, with surveys looking at energy aspects [134], load balancing techniques [135], and VM allocation in the DC [136]. Two levels of scheduling can be identified in an SDHI architecture: scheduling resources within a DC based on HW resource disaggregation to realize logical servers (covered in this section) and scheduling within a logical server to execute the workload (addressed later in this paper).

Resource scheduling within a DC based on SDHI involves selecting resources based on their properties and physical &

topological information. A scheduling mechanism must take into account information about each individual resource along with DC-wide network & interconnection fabric information, such as latency and available bandwidth between resources. Properties of the interconnect fabric (specifically latency and available bandwidth) between the resources play a crucial role in scheduling a logical server. Workloads have varying requirements on the performance of the logical server, hence while scheduling resources for a logical server the profile of the workload (see Section V-D3) needs to be taken into consideration to ensure that the requirements of the workload that run inside the logical servers are fulfilled. Scheduling mechanisms can be complex when the mechanism sees and selects every individual component or simpler when the mechanism only sees and selects the pool from which a certain resource should be allocated. In the latter case, each pool would be responsible for its internal scheduling, allowing individual optimization mechanisms per pool, thus reducing the complexity of the core scheduling mechanism. Moreover, scheduling operations should not be static, hence optimization should be possible after allocations have been made. Some of the aspects that the scheduling mechanisms should take into account are power consumption, resource defragmentation (i.e., avoiding composing logical systems of resources that are too scattered), performance, and optimized utilization (considering aspects such a component's lifetime to allow, for example, overclocking and overprovisioning in a smarter way).

While the benefits of SDHI and disaggregated environments have been extensively advocated, one needs to understand how to "carefully" foster them. When setting up a DC environment, dimensioning and distribution of resources come hand in hand. In a highly flexible environment with physical resource distribution, finding the optimal physical distribution/location of resources is even more important. For example, distributed resources bring an associated networking cost, making it necessary to find a balance between benefit and cost. For example, separation of memory and CPU by long distances might be possible, but an expensive interconnect technology would eat into the potential benefits of deploying such technology. B. Abali et al. [96] have done some assessment of the cost of memory disaggregation.

Another challenge is to understand and minimize the risk of distribution of single failures when placing CPU and memory pools in separate chassis. For example, if a chassis hosting a CPU pool where several (logical) hosts are running has a power failure, then all of these hosts will fail. As a result, HA requirements could be a constraint on the resource composition mechanism. Thus there are opportunities to realize different HA methods for an SDHI other than those used in current SDI.

Finally, it is important to highlight that the initial resource distribution is performed based on workload forecasting. However, it is extremely likely that workload patterns change over time, hence performing physical re-distribution of resources will be beneficial (e.g., moving a certain memory pool from one logical server to another). Having suitable mechanisms in place to optimize these re-distributions is therefore required.

3) Monitoring and Analytics: In a SDHI that aims to be as automated as possible, monitoring, fault prediction, and anomaly detection play a fundamental role. These mechanisms have the potential to take the resiliency and robustness of the infrastructure to a level unseen in current SDI architectures. Unlike today's infrastructures, these mechanisms should address both physical resources and logically composed resources. Therefore, the ability to retrieve near real-time information from the infrastructure is crucial. While retrieving physical information might be seen as a relatively straightforward task, retrieving information from logical components and relating this information to the information from physical components is not. For example, if multiple hosts share a certain block of physical memory or part of a CPU, this raises the question of how the system can collect monitoring information not only regarding the physical component, but also information from the portions of each component associated with each logical host.

Moreover, predictive models need to be developed so that the infrastructure can act proactively. For example, if one detects that a certain component is behaving abnormally (e.g., at a high temperature, which can indicate that a failure is likely), then one could trigger the migration of the associated logical servers to another component. However, not everything can be predicted; therefore, fault detection mechanisms will still be needed.

4) Hardware Re-planning and Setup: Given the complexity and dynamics of cloud solutions (especially in a disaggregated architecture), J. M. Soares says that cloud providers should adopt a modern and structured way to operate [191]. According to T. Thanakornworakij et al. success lies in careful planning [63] and continuous execution of optimization processes. This suggests that a new degree of agility, speed, and flexibility is needed to realize the best operational practices and to offer the most competitive pricing. Using a disaggregated architecture gives cloud operators a new dimension to exploit, thus making it possible to optimize their operations by reshuffling resources of various types as needed to increase resource utilization. Such an approach can help ensure that the infrastructure and its operation remain suitable for those workloads running on top of them.

This approach allows proactive DC hardware planning using run-time performance parameters extracted via analytics. By integrating a cost engine, operators can more agilely micro-plan their operational phases, thus supporting continuous replanning/re-shuffling of infrastructure resources and automatic cost reduction during the DC's lifetime. This approach can help reduce the TCO by enhancing resource utilization by using application-optimized hardware, thereby introducing greater transparency into the costs in the DC (e.g., see [60]).

Such a system might work as shown in Fig. 12. The DC automation platform continuously monitors the performance of the services and collects performance metrics, such as power consumption, network congestion, CPU utilization, communication delays, and response time. This monitoring data is analyzed by an analytics engine, based on pre-defined rules and thresholds. If the analysis detects the need for

changes in the system, such as the need for new hardware to support a consistently increasing workload, one or more events/alarms are generated and sent to the cost engine with the associated information. TCO will be calculated for various options and then the engine decides if immediate action is required or if the alarm should be ignored at this stage (this decision is based upon a consideration of the expenses of the system, service level agreements, or other policies). If there is a decision to act that involves purchasing new hardware, then the recommendations will be sent to a hardware dimensioning engine to calculate volume and type of the components to be purchased.

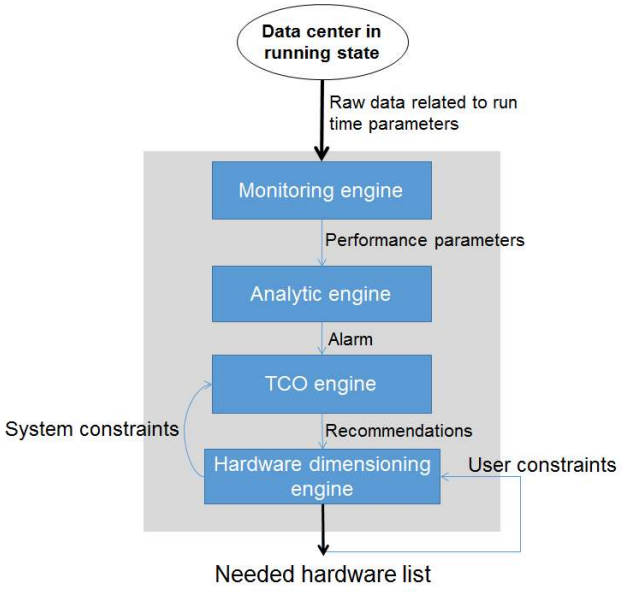

Fig. 12. Schematic overview of a DC automation platform's monitoring and planning modules.

The concept of a DC automation platform can be clarified using the following example use case. Assume that a DC is operating and the monitoring system detects a noticeable increase in the power consumption of some CPUs. This increased power consumption can be due to aging of the servers. Assuming two years of the server's lifetime, the CPU's efficiency has decreased [192], so more power is expended for the same workload. In this case, the analytics engine sends a notification to the TCO engine. The TCO engine calculates the cost of replacing the old servers (or possibly only old CPUs) with new ones versus the cost of energy that will be wasted without such a change. The TCO engine might take into consideration that hardware refreshment is already planned in two months time (assuming that the computing resources are replaced on some schedule, such as every 3 to 5 years). If so, the TCO engine might conclude that changing the hardware two months earlier than planned would decrease the TCO by saving sufficient energy to justify the cost. In this case, the TCO engine sends its recommendation to the hardware dimensioning engine to calculate the volume and type of new hardware to be purchased based on the current hardware's performance and the expected workload. It should be noted that the hardware dimensioning and selection in this step is more intelligent than the initial planning since for the DC the performance parameters related to the operating DC are available, hence this additional information will be used by the TCO engine when defining its proposal. Finally, the results will be presented to the user to make the final decision.

D. Platform Layer

Efficient use of SDHI requires the interplay of all the layers of the architecture (shown in Fig. 6). In this section, we look at some key challenges faced by the platform layer (specifically those regarding the OS and supporting services needed to run the application) in order to achieve the goals of SDHI and HW resource disaggregation.

1) Operating System: Resource disaggregation is a relatively a new concept and its impact on OS and application design and functionality will be of great importance with advancements in SHDI technology. From an OS and execution environment perspective, the first impact of a new HW architecture is felt at the OS level. In this case, the OS needs to understand how to cope with the impact of disaggregation (e.g. regarding CPU scheduling, memory/storage usage, and potential scalability issues). Few efforts have been made in this area, except for some initiatives (e.g., [193]). Thus we present some of the challenges that the OS will have to cope with to run over disaggregated HW.

From an OS perspective, the three most important design aspects to consider while managing a logical server composed of disaggregated resources are:

1. SDHI Interface: Unlike in the traditional situation where the OS is aware of types and quantities of the resources, in a disaggregated HW scenario an OS needs to (directly or indirectly) interface with the SDHI to allocate, free, and manage resources. The response time of this interface will have an impact on the application performance. For example, smooth resource scaling and/or migration should be possible, but would be difficult to implement without support from OSs (or a hypervisor, as described in section V-D2).

2. Hot (un)Plug of resources: It is critically important that the OS supports and manages smooth and non-disruptive Hot plug and unplug of resources, thus enabling the logical server to scale up and down based on the demand of the workload. CPU hot-plug is implemented in the Linux Kernel and allows one to add/remove CPUs on demand [143]. Removing a CPU starts by switching it to offline, causing all the tasks, interrupts, and timers to be migrated to another CPU. The case of memory hot-plug can be divided into two phases: (1) physical memory hot-plug phase (for physically adding/removing DIMMs) and (2) logical memory hot-plug phase (for changing the amount of allocated memory). Kernel memory offloading is supported starting with Linux kernel 3.8 with the limitation that the node must always have sufficient kernel memory [144]. Challenges also arise regarding memory removal while scaling down, such as how to select and free a logical host's memory slot with minimal impact on the OS and running applications. Today, the information required for smooth resource scaling can only be collected from *within* the OS, and therefore requires the participation of the OS in scaling down memory.

3. Abstraction: The OS needs to abstract away the fact that it is managing disaggregated resources from the application running on top of it. This will be crucial for many current state of the art applications. The OS will hide the complexity of managing disaggregated resources from the application.

4. Scheduling: As noted earlier, scheduling in a resource disaggregated DC is done at two levels: (1) by the SDHI manager when composing a logical host from slices of resources and (2) by the OS when executing a workload within these logical hosts. Given a one-to-one mapping between the workload and a logical host, the execution environment scheduler takes most of the responsibility for workload scheduling, including distribution of the workload's tasks and execution of each task's jobs. As a result the OS and its corresponding execution environment can be modified to support scheduling that is aware of resource disaggregation and the notion of remote vs. local memory or even different memory types. For example, the OS needs to be aware of the inter-connectivity characteristics (latency and bandwidth) of the composed logical server to make smart scheduling decisions. The OS needs to monitor (or be supplied with) the current state of various inter-connectivity characteristics in (near) real-time to make meaningful scheduling decisions. Smart scheduling in a resource disaggregated DC creates opportunities to increase system performance through increased spatial and temporal coherency. A high degree of spatial and temporal coherence results in greater data locality, hence processors will have the correct data at the correct time, reducing the gap between CPU processing speed and memory access latency.

While the OS can provide the abstraction of the disaggregated resources to the application, exposing the SDHI interface to the application can provide certain advantages that an application can leverage. By exposing the SDHI interface to applications, they can scale up and down their resource needs in a finer granularity given accurate and rapid mechanisms, hence facilitating efficient use of resources in the DC. It is essential that the SDHI can enforce application-level access control so that applications cannot demand or allocate resources that they are not authorized to use. The application can also use the SDHI interface to realize better resource sharing with other applications; one such use case (noted earlier) is the possibility of sharing RAM between applications to exchange data between applications thus eliminating the need for interprocess communication. But this exposure comes at the cost of increases in the complexity of the application, while legacy applications cannot adapt to using the SDHI interface (making the abstraction layer provided by the OS essential) (see Section V-D5).

2) Resource sharing and Hypervisor: Virtualization introduces a software abstraction layer (i.e., hypervisor) that emulates the abstract behavior of physical entities' attributes with software. The term virtualization has been used in two different contexts in the literature: network virtualization [17, 139] and server virtualization [16] (see Fig. 13).

Virtualization was first proposed for computer systems and servers with the primary objective of providing an abstraction that enables resource consolidation, resource slicing, and

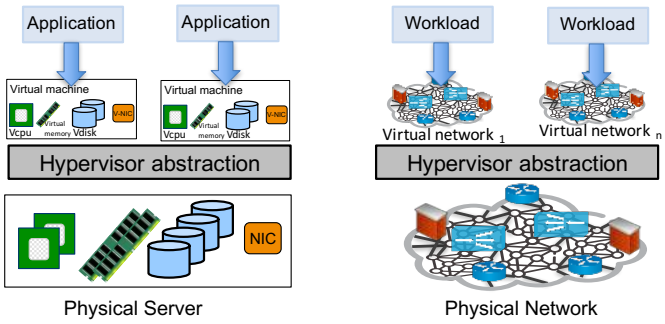

Fig. 13. Server virtualization versus Network virtualization.

multi-tenancy. In a server virtualization, the hypervisor layer emulates the attributes of the physical server, to realize a virtual machine (VM) with arbitrary sets of resources. Typically a new VM can be realized within a matter of seconds.

The success of server virtualization lead to speculation about network virtualization. In network virtualization, a network hypervisor realizes a physical network based upon a pool of transport capacity. As a result, the physical network's capacity can be sliced (on-demand) into different virtual network layouts (i.e., including networking entities performing switching, routing, and load balancing) in such a way that each virtual network can be treated separately from the underlying physical network hardware.

To realize resource sharing across different logical systems, the ability to both slice and aggregate resources is essential. In today's server-based architecture, resource sharing is frequently accomplished with the help of **server** virtualization and hypervisor software. However, in a software-defined resource disaggregated DC one should not be limited to the usage of legacy hypervisor-based technologies; but rather, one should exploit resource sharing in a far different manner. Potentially, DC providers could improve resource utilization through resource sharing (by avoiding resource stranding and fragmentation), while at the same time, avoiding the performance overhead introduced by software-based virtualization.

Fig. 14 shows four different examples of how resource slicing, sharing, and aggregation could be realized in an SDHI. In the case of A, a dedicated physical unit is provided for a host, hence no slicing or sharing occurs. Case B shows an example in which resource units can be sliced, with each slice assigned to different hosts as dedicated logical units. Case C shows the case of overprovisioning where, in addition to slicing, resource sharing is occurring. Finally, case D shows an example where multiple resource units are aggregated and provided to the host as a single logical resource. Beyond these cases, it should be noted that other approaches that are hybrids of these are possible. Moreover, virtualization technologies based on hypervisors could be applied at the logical host level.

Realizing resource slicing and sharing at the HW level (see Section V-A) could bring typical hypervisor functions to the HW, thus eliminating or reducing the overhead of current hypervisors. However, the question remains whether

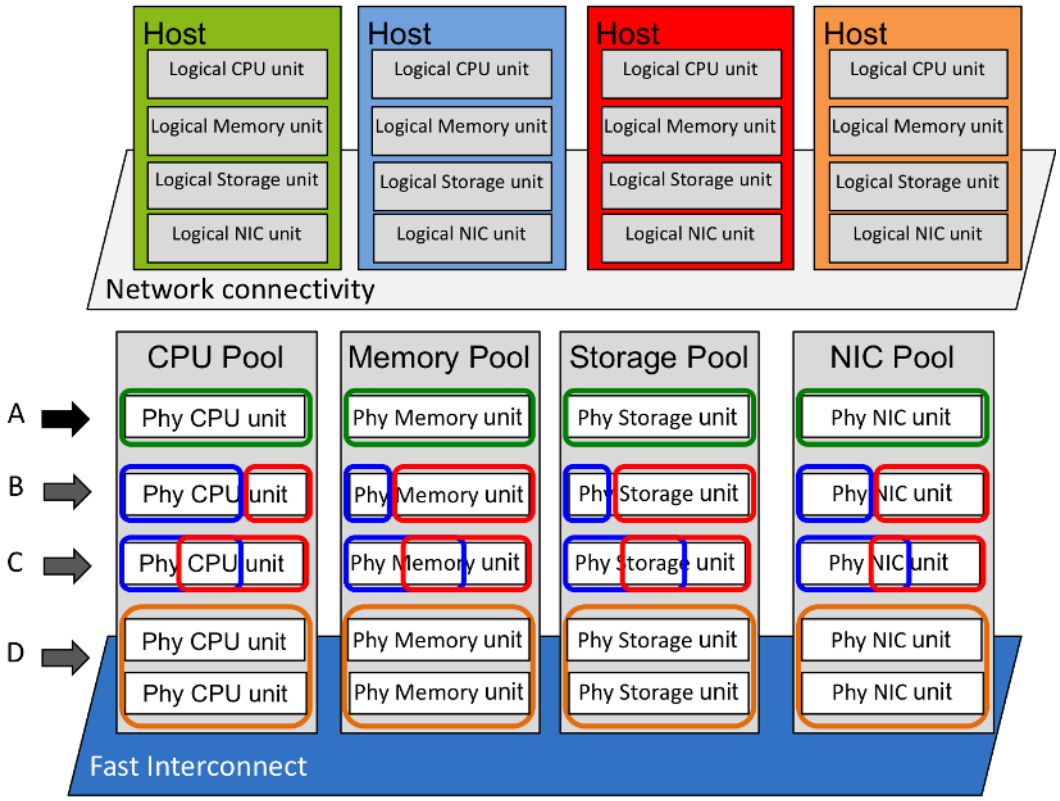

Fig. 14. Resource slicing and sharing (adapted from [58])

hypervisors will still be required. To address this issue, we need to consider the dependency on hypervisors and the essential functions that hypervisors provide, such as HW abstraction. Therefore, it is important to understand what role hypervisors will play and how they should evolve in conjunction with this new architecture.

Within an SDHI (i.e., based on resource disaggregation), when an entire CPU is not assigned to a logical server, a minimal hypervisor abstraction is still required to set up, partition the HW devices, and start and stop a given logical host. Monitoring of logical hosts is another function that could easily be done by a hypervisor, although such monitoring can be implemented in other ways (e.g., [194]) (see Section V-C3).

Arrakis [145] describes the advantages of removing the Linux kernel from the data path of IO and using the kernel only for the control path. Earlier work by Massalin [142], synthesized kernel code using just in time techniques to optimize kernel processing but still required going through the kernel. This concept can be extended to hypervisors at a higher granularity, by removing some aspects of the hypervisor from the data path. An application running under a minimal hypervisor greatly benefits from a reduction in expensive hypervisor calls. When it comes to processor resources, after a hypervisor has allocated specific CPU cores to the logical host, it does not need to participate in the subsequent scheduling of processes on these cores. Disaggregation has the potential to remove page fault handling functionality from the hypervisor and move it to the application, hence removing the hypervisor from the data path. Regarding networking, SR-IOV and Multi-Root I/O Virtualization (MR-IOV) can be used to directly talk to NICs using single/multiple queues after the HW has been initialized by the hypervisor. Similarly, storage devices supporting an NVM Express (NVMe) [146] interface can provide access to a storage namespace which can be given to a logical host, therefore bypassing the kernel (although this requires some security enhancements to prevent other logical machines from reading/writing from/to the same namespace). Furthermore, logical hosts using SDS or controller-based storage can work without any hypervisor support in the control path.

Removing some of the functions from the hypervisor may come at some cost. For example, assigning dedicated memory to a logical host using disaggregated memory could end up using more memory, as some features, such as kernel same page merge (KSM) [141], cannot be easily deployed. While it might still be possible to realize KSM functionality at a memory blade level, that may add more complexity to the memory blade hardware. Sharing memory (see section V-D2) between logical hosts could also become complicated in the absence of a hypervisor. Furthermore, with the removal of hypervisors, the ability to oversubscribe and to control devices that do not support resource slicing and sharing may be lost. In the absence of a hypervisor, logical hosts need a variety of hardware drivers to support the underlying hardware, and this could be challenging in a cloud environment with a diverse set of devices.

When it comes CPU slicing, while controls exist in today's processors to support cache partitioning and memory

bandwidth partitioning, methods to control partitioning of the bandwidth of interconnects (e.g., PCIe and SATA) are lacking. It might be easier to control the bandwidth from device to CPU if a single device is attached to the bus, but in the absence of a hypervisor this might require more complex queue management logic on the CPU. Functions crucial for the efficient use of CPU resources, such as on-demand allocation/removal of CPU slices (e.g., adding an extra core or giving up one), will also become more complicated without a hypervisor, hence requiring new interfaces or explicit logic in the applications, OS, or perhaps in the HW.

3) Sharing Resources Across Platforms: One of the key factors that underlies the resource efficiency of the disaggregated hardware architecture is that hardware resources can be dynamically shared among multiple logical servers, both in time (i.e., a hardware resource that was part of a logical server at one time can be moved to another logical server) and space (i.e., sharing of a hardware device across two logical servers at the same time, often by using hardware-implemented resource slicing functions). The challenges associated with these are discussed in Section V-D2. However, this ability on its own is insufficient to achieve resource efficiency; rather we need to fully exploit the software stack that runs on the logical servers to realize this goal. The following are some of the challenges faced by the software stack to fully exploit resource sharing to achieve a high level of resource utilization.

Demand Prediction/ Application Profiling: A high level of resource utilization can only be achieved when the *correct* amount of each type of resource is allocated to each application. However, knowing the exact amount of each resource needed by an application at any given time is not an easy task. First, it requires developing a model that can predict the demands/requests that are going to be made to the application itself (i.e., demand prediction). Then, given a level of demand (requests to the application), a model is needed that can predict the amount, type, and configuration of the logical host(s) needed to efficiently support this application's execution to meet these demands (i.e., application profiling). Although there exists considerable work on demand prediction [195–197], it unclear which approach is suitable for a resource disaggregated DC environment. While application profiling for server-based SDI has been extensively investigated, work on dynamically constructed logical servers (built from disaggregated hardware resources) is still in its infancy.

Cross-Platform Resource Sharing The traditional way to deploy software platforms is in so-called silos where each platform is allocated its own set of hardware resources which it utilizes, independently of the other platforms that are deployed in parallel. This limits the potential resource utilization since resources unused in one platform remain unused. Apache Mesos [83] is a cluster resource manager that introduced a new approach that enables resource sharing across multiple platforms, thereby increasing resource efficiency of those resources it manages. However, Mesos's approach assumes traditional server architectures and that platforms are adapted to run on a Mesos-managed cluster of resources. To achieve the high resource utilization promised by SDHI, a Mesos-like resource management system for disaggregated resources is essential.

4) Software Platforms for Disaggregated HW: A disaggregated HW architecture brings with it opportunities that allow existing software platforms to change how they do things to improve their efficiency, performance, and scalability. The following discuss just a few of these possibilities.

IaaS-Platforms Traditional IaaS platforms such as OpenStack use hypervisor-based virtualization technologies to slice up server resources and present them as VMs. However, given that resource disaggregation and SDHI can compose a logical server, its performance and efficiency can be improved by eliminating the hypervisor and running what would have been the 'VM' directly on the logically composed server. The RedFish driver for OpenStack's Ironic project [138] is the first step in this direction. Moreover, the OpenStack's Valence project [137] aims to provide a complete integration with an SDHI by being the integration point between OpenStack and SDHI management layer. More concretely, Valence considers Redfish (and Intel RSD specifications) as the appropriate integration point.

BigData Platforms Faster access to data in big data applications is the key to improving their performance. For example, Apache Spark achieves orders of magnitude improvement of application execution times compared to MapReduce because Spark uses Resilient Distributed Datasets (RDD) that reside in memory and hence can be accessed with low delay. Similar performance improvements can be expected by making use of shared memory in disaggregated hardware systems. Specifically, one can imagine a setting where the producers and consumers of an intermediate computation (e.g., mappers and reducers in MapReduce) share a memory area such that the producers write the result of their computation into the shared memory, while the consumers read and process the data from this shared memory. One may also consider directly sharing disks on which data is stored among workers of a big data platform, eliminating the need to duplicate the data at several places and saving on storage resources. In both cases, there is a trade-off between faster access and a more complex interaction between the various entities.

5) Application development: With SDHI, associated HW capabilities and features, and new classes of HW devices that continue to be added to new DCs (e.g., NVM and FPGA), it is important to understand how these capabilities affect applications, and how applications can take full advantage of this HW. There are two approaches. The first approach is to hide these details in a layer below the application, e.g., in the OS or even SDHI, thus applications will continue to be developed as they are today, while the HW or OS deal with the effects of disaggregation and new classes of HW devices. This approach is particularly interesting when running legacy

applications. The second approach is to make applications aware of disaggregation. This would enable applications to distinguish and use different types of resources (such as remote vs. local memory and RAM vs. NVM) while adapting their behavior to the HW. This would likely result in the best application performance, but comes at the cost of extra complexity in the application development process.

VI. Research Initiatives and Tools

This section provides an overview of some of the most prominent research initiatives in the field of SDHI. Moreover, it also refers to a set of tools that can be used to further explore the area of HW disaggregation and SDHI.

A. Research projects

The SDHI concept has captured the interest of both industry and the research community. Examples of the technology push driven by collaborative research projects between industry and academia include dRedBox [157], M2DC (Modular Microserver Data Centre) [158], and FireBox [156].

Of these, dRedBox aims to specify, design, and prototype a DC system based on a software-defined and HW disaggregated architecture where SoC based microservers, memory modules, and accelerators are placed in separate chassis/server trays and interconnected via a high-speed, low-latency opto-electronic fabric. Additionally, the project aims to provide management software to configure the infrastructure, focusing on resource/power management aspects.

Similar to the dRedBox project, FireBox aims to develop a new system architecture for DCs based on HW disaggregation. It mainly seeks to produce: custom data center SoCs, distributed simulation tools for large-scale machines, and system software for FireBox-like disaggregated environments. Moreover, special effort is made to integrate NVM into the architecture via a low-latency high-bandwidth optical switch.

The M2DC project also pursues the vision of a software-defined DC environment. However, its focus is on creating a high-density and software-definable server architecture based on microserver modules. Without aiming for a fully disaggregated environment, M2DC intends to allow a seamless combination of heterogeneous microservers within one enclosure. In addition to direct communication between microservers through a dedicated high-bandwidth and low latency communication network, the system also supports connections to storage or I/O extensions, allowing easy integration of PCIe-based extension cards such as general-purpose computing on graphics processing units (GPGPUs) or storage subsystems. Similar to dRedBox, M2DC also aims to provide middleware software to allow the infrastructure to run as efficiently as possible.

B. Emulation and simulation of resource disaggregation

SDHI and HW resource disaggregation is still in its infancy and hardware is not readily available for research. A resource disaggregated system would include a wide variety of computing, storage, memory, GPUs, FPGAs, interconnects, and peripherals as well as systems to control them. Unfortunately, it is not easy to emulate the complete system. However, a piecemeal approach is possible. FPGAs, hardware simulators, and QEMU are some of the tools used by various communities to develop SDHI systems. Table VII summarizes these tools and the following paragraphs describe some of these tools.

Datacenter-In-A-Box at Low Cost (DIABLO) [47, 156] is a DC simulator often described as a wind tunnel for DCs. It utilizes FPGAs and is capable of simulating on the order of 10,000 servers and 1000 switches. In this simulator, parameters such as link speed, latency, and CPU clock speed can be configured. DIABLO is built using 65 nm Xilinx Virtex FPGAs, supports SPARC v8 instructions, and is capable of running a Linux OS. DIABLO2 [155] is the second generation of DIABLO and plans to use 14nm FPGAs and 100 Gbps interconnects. The goal for this environment is to facilitate research on rack level optimizations.

Firesim [156] is another simulator being developed by the University of California at Berkeley's Architecture Research Lab. The project aims to enable cycle-accurate prototyping and benchmarking of new datacenter designs using public-cloud infrastructure. Firesim promises to allow users to evaluate the trade-offs between simulation accuracy and performance at a component granularity. Currently, Firesim can be realized on Amazon Web Services [200].

QEMU [140] provides an excellent environment for emulating various hardware resources, system behaviors, modeling new instructions, and inserting delays. The software is widely used and it is easy to get support from the community. QEMU can be used for emulating a single system or several systems. However, it is difficult to realistically emulate low latency or high throughput traffic. This emulator can be useful in understanding changes in an application's behavior in conjunction with hardware changes.

Maze [113] is a network emulation platform used by Microsoft in various research activities regarding rack scale systems. The focus of Maze is a rack-scale network fabric. Maze runs on a cluster of servers connected by a high-bandwidth switched network based on RDMA. The network fabric of the rack is emulated as a virtual network on top of a switched network. RDMA is capable of achieving 38Gbps over a 40 Gbps link. Maze can be used to implement/emulate new routing/transport protocols in user space. The emulated transport copies packets using RDMA from the sender to the receiver. Although Microsoft promised to open source Maze, currently no information about this is publicly available.

COTson [199] is a full functional simulator developed with the aim of providing fast and accurate evaluation of current and future computing systems with full software stack and hardware models for cluster-level systems. In COTSon fast functional emulators and timing models cooperate to improve simulation accuracy and speed in order to be able to simulate the full software stack. It incorporates a statistical sampling approach trading accuracy for speed where needed. By incorporating a robust interface between functional and timing domains, it is able to leverage existing simulators.

TABLE VII
TOOLS USED BY VARIOUS COMMUNITIES TO SIMULATE/EMULATE SDHI SYSTEMS AND HW RESOURCE DISAGGREGATION.

Ref	Name	Use	Availability	Comments
[140]	QEMU	A software platform for emulating various hardware resources.	Open Source	Difficult to realize nanosecond delays and high throughput. Many works in disaggregation use this approach, such as [24, 25, 198].
[113]	Maze	Platform for emulating rack scale network fabric used in several Microsoft studies.	Unavailable	It was previously announced on its webpage that it would be open sourced. Currently it is unavailable. Some design details can be gleaned from various papers.
[47, 156]	Diablo	FPGA accelerated simulators for enabling high scale cycle accurate simulations from UC Berkeley labs.	Unknown	Diablo2 [155] is a planned extension of this platform. Firesim is another simulator which allows rack scale designs to simulated. Firesim can be cost-effectively realized on AWS. A good deal of information about the simulator is available, although full details are not published.
[199]	COTson	Full System Simulator for Cluster-level Systems	Open Source	Provides a robust interface between functional and timing models to be able to leverage existing simulators. It is able to provide accuracy in the simulation where needed.
[198]	DiME	Disaggreaged Memory Simulator for evaluating application performance	Open Source	Using modifications to Linux kernel, the emulator is able to assess the impact of application performance while accessing remote memory. The user can evaluate application performance at various delays.
[154]	CoFluent	Co-simulation platform using a SysML based description with existing hardware.	Commercial	A commercial platform that can be used predict an application's performance for the given cluster configuration and to investigate "what if" scenarios.
[153]	Simics	A Full system simulator of targeted hardware.	Commercial	A commercial platform that can be useful in developing software when building new hardware.

The ability to adjust accuracy and speed, enables COTson to be used to evaluate all the way from microarchitecture level changes to evaluating customer requirements for clusters with performance/power and cost.

DiME [198] is a emulator for assessing the performance of applications in a disaggregated memory system. The emulator assumes a certain amount of memory is available locally. When remote memory pages are referenced they are swapped into the local memory. This allows the user to configure memory access latencies and inject these latencies. In its current implementation, DiME does not account for any queuing delays of interconnects or other system aspects, such as DRAM bandwidth.

CoFluent [154] is a commercial simulation platform that allows for co-simulation of a UML-based system description and the hardware used to implement the CPU, network, and memory resources. CoFluent predicts behavior by executing SysML (Systems Modeling Language) specifications and determines the performance of time constraints, bus transactions, memory accesses, power consumption, memory footprint, and cost. An application is used along with a functional model that together provide a refined virtual system model. A mapping is made between the functional model and the hardware resource model by allocating functions to processing units, data to storage units, and routing inter-processor data to physical links. CoFluent can predict an application's performance for a given cluster configuration and can be used to investigate "what if" scenarios or to test various design choices.

Simics [153, 201] is a commercial full system simulator in which the target hardware can be simulated. Simics can simulate systems ranging from a single component to multiple interconnected boards. Simics allows code compiled for the target hardware to be run on this virtual platform, hence it can be used when investigating system level issues.

As part of OpenStack Valance [137], a project to integrate rack scale architectures in OpenStack, a simulation capability is available for simulating rack scale node addition and removal. The exact capabilities of the Valance are currently unknown.

VII. CONCLUDING REMARKS

The IT industry has realized that to achieve increased cloud efficiency, some of the today's basic cloud infrastructure and SDI principles need to change. At the forefront of these (already ongoing) changes is the concept of SDHI, a concept that requires the cloud infrastructure to be re-architected in fundamental ways.

The journey towards SDHI is ongoing but its impact is already noticeable. The technological impact may seem to be confined to the HW and its management; however, the actual impact goes far beyond this ranging from the execution layer to the application layer. Although storage has already been disaggregated, challenges remain in how to realize decoupling of the remaining server components, such as memory, processors, and NICs. Further progress to a great extent, depends on HW vendors to support and add the required functionality into their HW components. We believe that over time increasing degrees of disaggregation will be supported by HW vendors. The effects of these changes will be ever more visible to the higher layers in the DC architecture. The article shows that memory disaggregation has received the greatest attention compared to other components and layers in SDHI. Despite this, memory disaggregation is still not mature. As a result, many questions remain to be answered, many independent efforts should be integrated, and additional efforts are required to realize a fully operational SDHI. After some of these problem have been addressed and technology has become more mature the impact of SDHI will be increasingly visible to the application developer and end users.

This paper provides the first consolidated survey of SDHI (and HW disaggregation), by presenting the SDHI concept, its business and technical opportunities, along with enabling technologies, architectures and protocols. Moreover, it continuously points out the challenges and issues that still remain to be addressed in order to realize the full potential of SDHI. This work provide a good foundation for researchers and practitioners who are interested in gaining insight into the SDHI concept and associated technologies and who want to understand the overall impact of SDHI and HW resource disaggregation on the entire cloud ecosystem.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful comments. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Q. Luo, W. Fang, J. Wu, and Q. Chen, "Reliable broadband wireless communication for high speed trains using baseband cloud," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, p. 285, Sep 2012.

[2] V. G. Nguyen, A. Brunstrom, K. J. Grinnemo, and J. Taheri, "SDN/NFV-Based Mobile Packet Core Network Architectures: A Survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1567–1602, thirdquarter 2017.

[3] F. Han, S. Zhao, L. Zhang, and J. Wu, "Survey of Strategies for Switching Off Base Stations in Heterogeneous Networks for Greener 5G Systems," *IEEE Access*, vol. 4, pp. 4959–4973, 2016.

[4] J. An, K. Yang, J. Wu, N. Ye, S. Guo, and Z. Liao, "Achieving sustainable ultra-dense heterogeneous networks for 5g," *IEEE Communications Magazine*, vol. 55, no. 12, pp. 84–90, DECEMBER 2017.

[5] J. Wu, "Green wireless communications: from concept to reality [industry perspectives]," *IEEE Wireless Communications*, vol. 19, no. 4, pp. 4–5, August 2012.

[6] A. Roozbeh, "Distributed Cloud and De-centralized Control Plane: A Proposal for Scalable Control Plane for 5G," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Dec 2015, pp. 348–353.

[7] S. Fu, H. Wen, J. Wu, and B. Wu, "Cross-Networks Energy Efficiency Tradeoff: From Wired Networks to Wireless Networks," *IEEE Access*, vol. 5, pp. 15–26, 2017.

[8] Z. Du, L. He, Y. Chen, Y. Xiao, P. Gao, and T. Wang, "Robot cloud: Bridging the power of robotics and cloud computing," *Future Generation Computer Systems*, vol. 74, no. Supplement C, pp. 337 – 348, 2017.

[9] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things," *Future Gener. Comput. Syst.*, vol. 56, no. C, pp. 684–700, Mar. 2016.

[10] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: A Performance Comparison," in *Proceedings of the 2015 IEEE International Conference on Cloud Engineering*, ser. IC2E '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 386–393.

[11] G. Kandiraju, H. Franke, M. D. Williams, M. Steinder, and S. M. Black, "Software Defined Infrastructures," *IBM J. Res. Dev.*, vol. 58, no. 2-3, Mar. 2014.

[12] K. Bakshi, "Microservices-based software architecture and approaches," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–8.

[13] A. Eivy, "Be Wary of the Economics of Serverless Cloud Computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, March 2017.

[14] Matt Watson, "What Is Function-as-a-Service? Serverless Architectures Are Here!" [Online]. Available: https://stackify.com/function-as-a-service-serverless-architecture/

[15] "Ericsson Introduces a Hyperscale Cloud Solution." [Online]. Available: https://www.ericsson.com/assets/local/news/2015/3/intel-ericsson-solution-brief.pdf

[16] R. Y. Ameen and A. Y. Hamo, "Survey of server virtualization," *CoRR*, vol. abs/1304.3557, 2013.

[17] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 655–685, Firstquarter 2016.

[18] R. Buyya, R. N. Calheiros, J. Son, A. V. Dastjerdi, and Y. Yoon, "Software-Defined Cloud Computing: Architectural Elements and Open Challenges," *CoRR*, vol. abs/1408.6891, 2014.

[19] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "Software defined cloud: Survey, system and evaluation," *Future Generation Computer Systems*, vol. 58, no. Supplement C, pp. 56 – 74, 2016.

[20] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "SDDC: A Software Defined Datacenter Experimental Framework," in *3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 189–194.

[21] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk and A. Rindos, "SDStorage: A Software Defined Storage Experimental Framework," in *IEEE International Conference on Cloud Engineering*, March 2015, pp. 341–346.

[22] P. Yasrebi, S. Bemby, H. Bannazadeh, and A. Leon-Garcia, *VNF Service Chaining on SAVI SDI*. Cham: Springer International Publishing, 2015, pp. 11–17.

[23] Industry Specification Group for NFV, "Network Functions Virtualisation," http://www.etsi.org/technologies-clusters/technologies/nfv.

[24] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network Support for Resource Disaggregation in Next-generation Datacenters," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, ser. HotNets-XII. New York, NY, USA: ACM, 2013, pp. 10:1–10:7.

[25] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network Requirements for Resource Disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. GA: USENIX Association, 2016, pp. 249–264.

[26] J. Shuja, K. Bilal, S. A. Madani, and S. U. Khan, "Data center energy efficient resource scheduling," *Cluster Computing*, vol. 17, no. 4, pp. 1265–1277, 2014.

[27] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware Cluster Management," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 127–144, Feb. 2014.

[28] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Dec. 2008.

[29] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 275–287, Mar. 2007.

[30] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[31] R. Figueiredo, P. A. Dinda, and J. Fortes, "Guest Editors' Introduction: Resource Virtualization Renaissance," *Computer*, vol. 38, no. 5, pp. 28–31, May 2005.

[32] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Mallick, "Xen 3.0 and the Art of Virtualization," in *Proceedings of the 2005 Ottawa Linux Symposium*, Jul. 2005.

[33] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, May 2005.

[34] I. Habib, "Virtualization with KVM," *Linux J.*, vol. 2008, no. 166, Feb. 2008.

[35] G. Tong, H. Jin, X. Xie, W. Cao, and P. Yuan, "Measuring and Analyzing CPU Overhead of Virtualization System," in *Services Computing Conference (APSCC), IEEE Asia-Pacific*, Dec 2011, pp. 243–250.

[36] J. Li, Q. Wang, D. Jayasinghe, J. Park, T. Zhu, and C. Pu, "Performance Overhead among Three Hypervisors: An Experimental Study Using Hadoop Benchmarks," in *IEEE International Congress on Big Data*, June 2013, pp. 9–16.

[37] N. Regola and J. C. Ducom, "Recommendations for Virtualization Technologies in High Performance Computing," in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2010, pp. 409–416.

[38] T. Harris, "Hardware Trends: Challenges and Opportunities in Distributed Computing," *SIGACT News*, vol. 46, no. 2, pp. 89–95, Jun. 2015.

[39] E. Hooper, "Under the Hood of Intel Rack Scale Architecture." [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-rack-scale-architecture-under-the-hood-video.html

[40] "Intel Rack Scale Design." [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html

[41] "Open Compute Project," http://www.opencompute.org/.

[42] "Hewlett Packard Labs - The Machine," https://www.labs.hpe.com/the-machine.

[43] "Microsoft Rack-scale Computing," http://research.microsoft.com/en-us/projects/rackscale/.

[44] Hewlett-Packard Development Company, L.P., "HP Moonshot System The world's first software defined servers," HP, Tech. Rep., April 2013, (Accessed: 2017-12-02).

[45] "SeaMicro SM15000 Fabric Compute Systems." http://bit.ly/1hQepIh.

[46] "Boston Viridis Data Sheet." http://bit.ly/1fBnsQ9.

[47] K. Asanovic, "FireBox: A Hardware Building Block for 2020 Warehouse-Scale Computers," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*. Santa Clara, CA: USENIX, 2014.

[48] S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper, S. Legtchenko, A. Ogus, E. Peterson, and A. Rowstron, "Pelican: A Building Block for Exascale Cold Data Storage," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 351–365.

[49] A. Daglis, S. Novaković, E. Bugnion, B. Falsafi, and B. Grot, "Manycore Network Interfaces for In-memory Rack-scale Computing," *SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 567–579, Jun. 2015.

[50] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-out NUMA," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 3–18, Feb. 2014.

[51] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24.

[52] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends, "Rack-scale disaggregated cloud data centers: The dReDBox project vision," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 690–695.

[53] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.

[54] GigPeak, Inc., "GigPeak Announces Record Quarter Shipment of ICs for Data Center Applications, and Sampling of SR and LR PAM4 IC Chipsets," GigPeak, Tech. Rep., Sep. 2016. [Online]. Available: http://www.businesswire.com/news/home/20160915005508/en/GigPeak-Announces-Record-Quarter-Shipment-ICs-Data

[55] Gartner, Inc., "Gartner Says Worldwide IT Spending Is Forecast to Be Flat in 2016," Gartner, Tech. Rep., Jul. 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3368517

[56] N. C. Luong, P. Wang, D. Niyato, Y. Wen, and Z. Han, "Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 954–1001, Secondquarter 2017.

[57] M. Liaqat, V. Chang, A. Gani, S. H. A. Hamid, M. Toseef, U. Shoaib, and R. L. Ali, "Federated cloud resource management: Review and discussion," *Journal of Network and Computer Applications*, vol. 77, no. Supplement C, pp. 87 – 105, 2017.

[58] Ericsson AB, "Hyperscale cloud - reimagining data centers from hardware to application," Ericsson White paper, Tech. Rep. 284 23-3289 Uen, May 2016. [Online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-hyperscale-cloud.pdf

[59] J. Kozhipurath, "Cloud Service Costing Challenges," in *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Oct 2012, pp. 1–6.

[60] M. Mahloo, J. Soares, and A. Roozbeh, "Techno-economic framework for cloud infrastructure: a cost study of resource disaggregation," in *2017 IEEE/ACM Federated Conference on Computer Science and Information Systems*, September 2017.

[61] M. Pawlish, A. S. Varde, and S. A. Robila, "Analyzing utilization rates in data centers for optimizing energy management," in *2012 International Green Computing Conference (IGCC)*, June 2012, pp. 1–6.

[62] Mainstay, "An Economic Study of the Hyperscale Data Center," Mainstay White paper, Tech. Rep., Jan. 2016. [Online]. Available: cloudpages.ericsson.com/hubfs/Content-Offers/Economic-Study-Hyperscale-Datacenter.pdf

[63] T. Thanakornworakij, R. Nassar, C. B. Leangsuksun, and M. Paun, "An Economic Model for Maximizing Profit of a Cloud Service Provider," in *2012 Seventh International Conference on Availability, Reliability and Security*, Aug 2012, pp. 274–279.

[64] Oracle Corporation, "Reducing costs through better server utilization," Oracle White paper, Tech. Rep., Apr. 2010. [Online]. Available: http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/reducing-costs-wp-075962.pdf

[65] J. E. Moreira and J. Karidis, "The Case for Full-Throttle Computing: An Alternative Datacenter Design Strategy," *IEEE Micro*, vol. 30, no. 4, pp. 25–28, July 2010.

[66] T. Deguchi, Y. Taniguchi, G. Hasegawa, Y. Nakamura, N. Ukita, K. Matsuda, and M. Matsuoka, "Impact of workload assignment on power consumption in software-defined data center infrastructure," in *2014 IEEE 3rd International*

Conference on Cloud Networking (CloudNet), Oct 2014, pp. 420–425.

[67] Y. Taniguchi, K. Suganuma, T. Deguchi, G. Hasegawa, Y. Nakamura, N. Ukita, N. Aizawa, K. Shibata, K. Matsuda, and M. Matsuoka, "Tandem Equipment Arranged Architecture with Exhaust Heat Reuse System for Software-Defined Data Center Infrastructure," *IEEE Transactions on Cloud Computing*, vol. 5, no. 2, pp. 182–192, April 2017.

[68] M. H. Rehmani and M. Erol Kantarci and A. Rachedi and M. Radenkovic and M. Reisslein, "IEEE Access Special Section Editorial Smart Grids: a Hub of Interdisciplinary Research," *IEEE Access*, vol. 3, pp. 3114–3118, 2015.

[69] M. Husain Rehmani and M. Reisslein and A. Rachedi and M. Erol-Kantarci and M. Radenkovic, "Guest Editorial Special Section on Smart Grid and Renewable Energy Resources: Information and Communication Technologies With Industry Perspective," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3119–3123, Dec 2017.

[70] A. A. Khan and M. H. Rehmani and M. Reisslein, "Requirements, Design Challenges, and Review of Routing and MAC Protocols for CR-Based Smart Grid Systems," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 206–215, May 2017.

[71] J. Wu, S. Guo, J. Li, and D. Zeng, "Big Data Meet Green Challenges: Greening Big Data," *IEEE Systems Journal*, vol. 10, no. 3, pp. 873–887, Sept 2016.

[72] "OpenStack Open Source Cloud Computing Software," https://www.openstack.org/.

[73] Canonical Ltd., "Ubuntu Metal-as-a-Service," https://www.ubuntu.com/server/maas.

[74] "Kubernetes - Production-Grade Container Orchestration," https://kubernetes.io/.

[75] Intel, "Intelligent Platform Management Interface (IPMI)." [Online]. Available: https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html

[76] Hewlett Packard Enterprise Development LP, "HPE Integrated Lights Out," https://www.hpe.com/ca/en/servers/integrated-lights-out-ilo.html.

[77] Dell Inc., "iDRAC with Lifecycle Controller," http://www.dell.com/learn/us/en/15/solutions/integrated-dell-remote-access-controller-idrac.

[78] Distributed Management Task Force (DMTF), "Redfish API." [Online]. Available: https://www.dmtf.org/standards/redfish

[79] Amazon Web Services, Inc., "Amazon Elastic Compute Cloud (Amazon EC2) ." [Online]. Available: https://aws.amazon.com/ec2/

[80] Cloud Foundry, Inc, "Cloud Foundry Foundation." [Online]. Available: https://www.cloudfoundry.org/

[81] "IBM WebSphere Application Server," http://www.hypertransport.org/.

[82] "Adobe ColdFusion family," http://www.adobe.com/products/coldfusion-family.html.

[83] "Apache Mesos," http://mesos.apache.org/.

[84] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet Another Resource Negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16.

[85] 4Quant Ltd., "Big Image Analytics," http://4quant.com/.

[86] Intel, "Intel Rack Scale Design Reference Software," https://01.org/intelrsd.

[87] Telecom Infrastructure Project. [Online]. Available: https://telecominfraproject.com/

[88] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated Memory for Expansion and Sharing in Blade Servers," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 267–278, Jun. 2009.

[89] J. Haas and P. Vogt, "Fully-Buffered DIMM Technology Moves Enterprise Platforms to the Next Level," *Technology Intel Magazine*, p. 7, March 2005.

[90] A. Hadke, T. Benavides, S. J. B. Yoo, R. Amirtharajah, and V. Akella, "OCDIMM: Scaling the DRAM Memory Wall Using WDM Based Optical Interconnects," in *2008 16th IEEE Symposium on High Performance Interconnects*, Aug 2008, pp. 57–63.

[91] Intel, "Intel Optane Technology." [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html

[92] P. S. Rao and G. Porter, "Is Memory Disaggregation Feasible?: A Case Study with Spark SQL," in *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '16. New York, NY, USA: ACM, 2016, pp. 75–80.

[93] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, ser. HPCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–12.

[94] C.-C. Tu, C.-t. Lee, and T.-c. Chiueh, "Marlin: A Memory-based Rack Area Network," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '14. New York, NY, USA: ACM, 2014, pp. 125–136.

[95] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A Holistic Approach to Fast In-Memory Key-Value Storage," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 429–444.

[96] B. Abali, R. J. Eickemeyer, H. Franke, C. Li, and M. Taubenblatt, "Disaggregated and optically interconnected memory: when will it be cost effective?" *CoRR*, vol. abs/1503.01416, 2015.

[97] G. A. Gibson and R. Van Meter, "Network Attached Storage Architecture," *Commun. ACM*, vol. 43, no. 11, pp. 37–45, Nov. 2000.

[98] J. Maroney, S. Astarabadi, and S. Phong, "I/O card architecture based on a common controller," Feb 2017, US Patent 9,582,453.

[99] H. Dewan and R. C. Hansdah, "A survey of cloud storage facilities," in *2011 IEEE World Congress on Services*, July 2011, pp. 224–231.

[100] Z. Ou, Z. H. Hwang, F. Chen, R. Wang, and A. Yla-jaaski, "Is Cloud Storage Ready? A Comprehensive Study of IP-Based Storage Systems," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Dec 2015, pp. 1–10.

[101] Ding, Ruiquan, Hu, Xiao, Chen, Guofeng, Xiao, Zhiwen, Zhang, Jiajun, Liu, Chao, Wang, Jian, Zhou, Huan, and Zhang, Jun, "Rack-Scale Storage Fabric: A Practical Way to Build Best-Fit Infrastructure for High-Performance Data Processing," *MATEC Web Conf.*, vol. 76, 2016.

[102] A. Klimovic, C. Kozyrakis, E. Thereska, B. John, and S. Kumar, "Flash Storage Disaggregation," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: ACM, 2016, pp. 29:1–29:15.

[103] P. Fernando, S. Kannan, A. Gavrilovska, and K. Schwan, "Phoenix: Memory Speed HPC I/O with NVM," in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, Dec 2016, pp. 121–131.

[104] Intel Corporation, "Intel Rack Scale Architecture using Intel Ethernet Multi-host Controller FM10000 Family," Intel white paper, Tech. Rep., 2015. [Online]. Available: https://www.intel.com/content/www/us/en/ethernet-products/multi-host-controllers/rsa-using-fm10000-paper.html

[105] Mellanox, "Multi-Host Solutions." [Online]. Available: http://www.mellanox.com/page/products_dyn?product_family=210&mtag=multihost

[106] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout, "It's Time for Low Latency," in *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, ser. HotOS'13. Berkeley, CA, USA: USENIX Association, 2011, pp. 11–11.

[107] G. Liao and L. Bhuyan, "Performance Measurement of an Integrated NIC Architecture with 10GbE," in *2009 17th IEEE Symposium on High Performance Interconnects*, Aug 2009, pp. 52–59.

[108] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471 – 1480, 2012, communication Architectures for Scalable Systems.

[109] D. Thomson, A. Zilkie, J. Bowers, T. Komljenovic, G. Reed, L. Vivien, D. Marris-Morini, E. Cassan, L. Virot, J.-M. Fedeli, J.-M. Hartmann, J. Schmid, D.-X. Xu, F. Boeuf, P. O'Brien, G. Mashanovich, and M. Nedeljković, "Roadmap on silicon photonics," *Journal of Optics*, vol. 18, no. 7, pp. 1–20, June 2016.

[110] C. Batten, A. Joshi, J. Orcutt, C. Holzwarth, M. Popovic, J. Hoyt, F. Kartner, R. Ram, V. Stojanovic, and K. Asanovic, "Building Manycore Processor-to-DRAM Networks with Monolithic CMOS Silicon Photonics," *IEEE Micro*, vol. PP, no. 99, pp. 1–1, 2016.

[111] A. Chakraborty, E. Schenfeld, and D. D. Silva, "Switching Optically-Connected Memories in a Large-Scale System," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 727–738.

[112] J. Weiss, R. Dangel, J. Hofrichter, F. Horst, D. Jubin, N. Meier, A. L. Porta, and B. J. Offrein, "Optical interconnects for disaggregated resources in future datacenters," in *2014 The European Conference on Optical Communication (ECOC)*, Sept 2014, pp. 1–3.

[113] P. Costa, H. Ballani, K. Razavi, and I. Kash, "R2C2: A Network Stack for Rack-scale Computers," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 551–564, Aug. 2015.

[114] P. Costa, H. Ballani, and D. Narayanan, "Rethinking the Network Stack for Rack-scale Computers," in *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*. Philadelphia, PA: USENIX Association, 2014.

[115] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao, "XFabric: A Reconfigurable In-Rack Network for Rack-Scale Computers," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 15–29.

[116] M. Besta and T. Hoefler, "Slim Fly: A Cost Effective Low-diameter Network Topology," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 348–359.

[117] "An Introduction to the Intel QuickPath Interconnect," January 2009. [Online]. Available: https://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[118] Gen-Z. [Online]. Available: http://genzconsortium.org/

[119] OpenCAPI. [Online]. Available: http://opencapi.org/

[120] Cache Coherent Interconnect for Accelerators (CCIX). [Online]. Available: http://www.ccixconsortium.com/

[121] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos, "A Survey on Data Center Networking for Cloud Computing," *Comput. Netw.*, vol. 91, no. C, pp. 528–547, Nov. 2015.

[122] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A Survey on Data Center Networking (DCN): Infrastructure and Operations," *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 640–656, Firstquarter 2017.

[123] H. Huang, S. Guo, J. Wu, and J. Li, "Service Chaining for Hybrid Network Function," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[124] H. Huang, S. Guo, J. Wu, and J. Li, "Green DataPath for TCAM-Based Software-Defined Networks," *IEEE Communications Magazine*, vol. 54, no. 11, pp. 194–201, November 2016.

[125] D. Kreutz, F. M. V. Ramos, P. E. Verassimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[126] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[127] A. Farshin and S. Sharifian, "A chaotic grey wolf controller allocator for Software Defined Mobile Network (SDMN) for 5th generation of cloud-based cellular systems (5G)," *Computer Communications*, vol. 108, no. Supplement C, pp. 94 – 109, 2017.

[128] A. Farshin and S. Sharifian, "MAP-SDN: a metaheuristic assignment and provisioning SDN framework for cloud datacenters," *The Journal of Supercomputing*, vol. 73, no. 9, pp. 4112–4136, Sep 2017.

[129] H. Huang, S. Guo, J. Wu, and J. Li, "Joint middlebox selection and routing for software-defined networking," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[130] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software Defined Optical Networks (SDONs): A Comprehensive Survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2738–2786, Fourthquarter 2016.

[131] 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 *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. CO: USENIX Association, Oct. 2014, pp. 49–65.

[132] 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," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. CO: USENIX Association, Oct. 2014, pp. 1–16.

[133] DMTF Scalable Platforms Management Forum (SPMF). [Online]. Available: https://telecominfraproject.com/

[134] Z. Usmani and S. Singh, "A Survey of Virtual Machine Placement Techniques in a Cloud Data Center," *Procedia Computer Science*, vol. 78, no. Supplement C, pp. 491 – 498, 2016.

[135] S. B. Shaw and A. K. Singh, "A survey on scheduling and load balancing techniques in cloud computing environment," in *2014 International Conference on Computer and Communication Technology (ICCCT)*, Sept 2014, pp. 87–95.

[136] Z. A. Mann, "Allocation of Virtual Machines in Cloud Data Centers - A Survey of Problem Models and Optimization Algorithms," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 11:1–11:34, Aug. 2015.

[137] Openstack Valence Project. [Online]. Available: https://wiki.openstack.org/wiki/Valence

[138] OpenStack Foundation, "RedFish Driver," https://docs.openstack.org/ironic/pike/admin/drivers/redfish.html.

[139] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366–380, Sept 2016.

[140] "QEMU Open Source Project," http://www.qemu-project.org/.

[141] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using KSM," in *Proceedings of the Linux Symposium*, 2009, pp. 19–28.

[142] H. Massalin, "Synthesis: An Efficient Implementation of Fundamental Operating System Services," Ph.D. dissertation, Columbia University, New York, NY, USA, 1992.

[143] S. Andrzej, R. Russell, S. Vaddagiri, A. Raj, and J. Schopp, "CPU hotplug in the Kernel," December 2016. [Online]. Available: https://www.kernel.org/doc/html/v4.11/core-api/cpu_hotplug.html

[144] Z. Yanfei, "Arrange hotpluggable memory as ZONE MOVABLE." [Online]. Available: https://lwn.net/Articles/568978/

[145] 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.*, vol. 33, no. 4, pp. 11:1–11:30, Nov. 2015.

[146] "NVM Express Working Group." [Online]. Available: http://www.nvmexpress.org

[147] M. Oikawa, A. Kawai, K. Nomura, K. Yasuoka, K. Yoshikawa, and T. Narumi, "DS-CUDA: A Middleware to Use Many GPUs in the Cloud Environment," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 1207–1214.

[148] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU Transparent Virtualization Component for High Performance Computing Clouds," in *Proceedings of the 16th International Euro-Par Conference on Parallel Processing: Part I*, ser. EuroPar'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 379–391.

[149] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, and P. Ranganathan, "GViM: GPU-accelerated Virtual Machines," in *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, ser. HPCVirt '09. New York, NY, USA: ACM, 2009, pp. 17–24.

[150] C. Reano, F. Silla, and J. Duato, "Enhancing the rCUDA Remote GPU Virtualization Framework: From a Prototype to a Production Solution," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2017, pp. 695–698.

[151] Tidalscale, "Software-Defined Servers," (Accessed 2018-12-27). [Online]. Available: https://www.tidalscale.com/

[152] ScaleMP, "Versatile SMP (vSMP) Architecture." [Online]. Available: http://www.scalemp.com/technology/versatile-smp-vsmp-architecture/

[153] "Wind River Simics." [Online]. Available: https://www.windriver.com/products/product-overviews/Wind-River-Simics_Product-Overview.pdf

[154] G. Xu, Z. Bian, I. Cremer, J. Gruher, and M. Riess, "Highly Accurate Simulations of Big-Data Clusters for System Planning and Optimization," Intel white paper, Tech. Rep., 2016. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/rsa-cofluent-paper.pdf

[155] "Diablo2," http://diablo.cs.berkeley.edu/about.html#diablo2.

[156] "Firesim," https://fires.im/.

[157] dRedBox Project, http://www.dredbox.eu/.

[158] A. Oleksiak, M. Kierzynka, W. Piatek, G. Agosta, A. Barenghi, C. Brandolese, W. Fornaciari, G. Pelosi, M. Cecowski, R. Plestenjak, J. inkelj, M. Porrmann, J. Hagemeyer, R. Griessl, J. Lachmair, M. Peykanu, L. Tigges, M. v. d. Berge, W. Christmann, S. Krupop, A. Carbon, L. Cudennec, T. Goubier, J.-M. Philippe, S. Rosinger, D. Schlitt, C. Pieper, C. Adeniyi-Jones, J. Setoain, L. Ceva, and U. Janssen, "M2dc modular microserver datacentre with heterogeneous hardware," *Microprocess. Microsyst.*, vol. 52, no. C, pp. 117–130, Jul. 2017.

[159] D. Mulnix, "Intel Xeon Processor Scalable Family Technical Overview," Sept 2017. [Online]. Available: https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview

[160] Gartner, "Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016," February 2017. [Online]. Available: http://www.gartner.com/newsroom/id/3598917

[161] S. Sureka, "The IoT Marathon: a Race for 100 billion connected things!" May 2016. [Online]. Available: http://community.comsoc.org/blogs/saurabhsureka/iot-marathon-race-100-billion-connected-things

[162] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010.

[163] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: ACM, 2012, pp. 7:1–7:13.

[164] C. H. Lam, "Storage Class Memory," in *2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology*, Nov 2010, pp. 1080–1083.

[165] R. F. Freitas and W. W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 439–447, July 2008.

[166] S. N. Mozaffari and S. Tragoudas and T. Haniotakis, "Fast march tests for defects in resistive memory," in *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH15)*, July 2015, pp. 88–93.

[167] S. N. Mozaffari, S. Tragoudas, and T. Haniotakis, "More Efficient Testing of Metal-Oxide Memristor-Based Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 6, pp. 1018–1029, June 2017.

[168] A. Roozbeh, J. Soares, and D. Turull, "Method and memory merging function for merging memory pages," 04 2017, wO Application 2017,067,569.

[169] A. Roozbeh and J. Soares and D. Turull, "Method and memory availability managing module for managing availability of memory pages," 06 2017, wO Application 2017,095,281.

[170] "InfiniBand," http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband.

[171] "Intel Omnipath," https://www.intel.com/content/www/us/en/high-performance-computing-fabrics/omni-path-architecture-fabric-overview.html.

[172] A. Dragojevic, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast Remote Memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, 2014, pp. 401–414.

[173] H. Montaner, F. Silla, H. Froning, and J. Duato, "$MEMSCALE^{TM}$: A Scalable Environment for Databases," in *2011 IEEE International Conference on High Performance Computing and Communications*, Sept 2011, pp. 339–346.

[174] "HyperTransport Technology Consortium," http://www.hypertransport.org/.

[175] R. Hou, T. Jiang, L. Zhang, P. Qi, J. Dong, H. Wang, X. Gu, and S. Zhang, "Cost effective data center servers," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 179–187.

[176] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient Memory Disaggregation with Infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 649–667.

[177] F. Li, S. Das, M. Syamala, and V. R. Narasayya, "Accelerating Relational Databases by Leveraging Remote Memory and RDMA," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY,

USA: ACM, 2016, pp. 355–370.
[178] A. Roozbeh, J. Soares, and D. Turull, "A method of live migration," Jan 2018, US Patent 9,875,057.
[179] S. Mittal, "A survey of techniques for architecting TLBs," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 10, pp. e4061–n/a, 2017, e4061 cpe.4061.
[180] "Storage Networking Industry Association," https://www.snia.org/.
[181] Cisco Systems, Inc., "Low-Latency Ethernet Solutions for High-Performance Computing," cisco, Tech. Rep., 2017. [Online]. Available: https://www.cisco.com/c/dam/en_us/solutions/industries/docs/education/ethernet-solutions-high-performance-computing-education.pdf
[182] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[183] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.
[184] V. Rao and E. Smith, "Facebooks new front-end server design delivers on performance without sucking up power," March 2016. [Online]. Available: https://code.facebook.com/posts/1711485769063510/facebook-s-new-front-end-server-design-delivers-on-performance-without-sucking-up-power/
[185] P.-H. Kamp, "You'Re Doing It Wrong," *Queue*, vol. 8, no. 6, pp. 20:20–20:27, Jun. 2010.
[186] N. Zilberman, A. W. Moore, and J. A. Crowcroft, "From photons to big-data applications: terminating terabits," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 374, no. 2062, 03 2016.
[187] G. P. Katsikas, G. Q. Maguire Jr., and D. Kostić, "Profiling and accelerating commodity NFV service chains with SCC," *Journal of Systems and Software*, vol. 127C, pp. 12–27, Feb. 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2017.01.005
[188] G. P. Katsikas, T. Barbette, D. Kostić, R. Steinert, and G. Q. Maguire Jr., "Metron: NFV Service Chains at the True Speed of the Underlying Hardware," in *15th USENIX Conference on Networked Systems Design and Implementation (NSDI 18)*, ser. NSDI'18. Renton, WA: USENIX Association, 2018, pp. 171–186.
[189] Gen-Z Consortium, "Gen-Z Overview." [Online]. Available: http://genzconsortium.org/wp-content/uploads/2016/11/Gen-Z-Overview-V1.pdf
[190] "YANG to Redfish Mapping Specification," October 2016. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/DSP0271_0.5.6.pdf
[191] J. M. Soares, "Cloud Disaggregation," *Ericsson Research Blog*, Mar 27, 2015. [Online]. Available: https://www.ericsson.com/research-blog/cloud/cloud-disaggregation/
[192] J. Keane and C. H. Kim, "An odomoeter for CPUs," *IEEE Spectrum*, vol. 48, no. 5, pp. 28–33, May 2011.
[193] Y. Shan, S. Hallymysore, Y. Huang, Y. Chen, and Y. Zhang, "Disaggregated Operating System," in *Proceedings of the 2017 Symposium on Cloud Computing*, ser. SoCC '17. New York, NY, USA: ACM, 2017, pp. 628–628.
[194] A. Roozbeh, A. Sefidcon, and G. Q. Maguire, "Resource Monitoring in a Network Embedded Cloud: An Extension to OSPF-TE," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, Dec 2013, pp. 139–146.
[195] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, P. P. Jayaraman, J. Kolodziej, P. Balaji, S. Zeadally, Q. M. Malluhi, N. Tziritas, A. Vishnu, S. U. Khan, and A. Zomaya, "A Survey and Taxonomy on Energy Efficient Resource Allocation Techniques for Cloud Computing Systems," *Computing*, vol. 98, no. 7, pp. 751–774, Jul. 2016.
[196] B. Jennings and R. Stadler, "Resource Management in Clouds: Survey and Research Challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, Jul 2015.
[197] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *2012 IEEE Network Operations and Management Symposium*, April 2012, pp. 1287–1294.
[198] D. Buragohain, A. Ghogare, T. Patel, M. Vutukuru, and P. Kulkarni, "DiME: A Performance Emulator for Disaggregated Memory Architectures," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, ser. APSys '17. New York, NY, USA: ACM, 2017, pp. 15:1–15:8.
[199] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "COTSon: Infrastructure for Full System Simulation," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 52–61, Jan. 2009.
[200] M. Champion, "Bringing Datacenter-Scale Hardware-Software Co-design to the Cloud with FireSim and Amazon EC2 F1 Instances," *AWS Compute Blog*, 25 OCT 2017. [Online]. Available: https://aws.amazon.com/blogs/compute/bringing-datacenter-scale-hardware-software-co-design-to-the-cloud-with-firesim-and-amazon-ec2-f1-instances/
[201] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb 2002.

APPENDIX
LIST OF ACRONYMS

AMB	Ambient Memory Buffer
API	application programing interface
BSS	Business Support Systems
CAPEX	capital expenditures
CCIX	Cache Coherent Interconnect for Accelerators
CPU	Central Processing Unit
DAS	Direct-attached Storage
DC	data center
DCB	DC Bridging
DCs	data centers
DDR	Double Data Rate
DIABLO	Datacenter-In-A-Box at Low Cost
DIMM	Dual Inline Memory Module
DMA	Direct Memory Access
DMI	Direct Media Interface
DMTF	Distributed Management Task Force
DRAM	Dynamic Random-Access Memory
ENI	Elastic Network Interface
FB-DIMM	Fully Buffered DIMM
FC	Fiber Channel
FPGA	Field Programmable Gate Array
GPGPU	General-purpose GPU
GPU	Graphics Processing Unit
HA	High Availability
HDI	Hardware-Defined Infrastructures
HDS	Hyperscale Data System
HPC	high-performance computing
HRDMA	Hardware based RDMA
HW	hardware
IaaS	Infrastructure as a Service
IoT	Internet of Things
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
iSCSI	Internet SCSI
IT	information technology
iWARP	internet Wide Area RDMA Protocol
KSM	kernel same page merge
MaaS	Metal as a Service
MR-IOV	Multi-Root I/O Virtualization
NAS	Network Attached Storage
NIC	Network Interface Card
NTB	Non-Transparent Bridged
NUMA	Non-uniform memory access
NVM	Non-Volatile Memory
NVMe	NVM Express
OCDIMM	Optically Connected DIMM
OCP	Open Compute Project
OpenCAPI	Open Coherent Accelerator Processor Interface
OPEX	operational expenditures
OS	operating system
OSS	Operations Support Systems
PaaS	Platform as a Service
PCIe	Peripheral Component Interconnect Express
PODM	Pod Manager
PSME	Pooled System Management Engine
QPI	Quick Path Interconnect
RAM	Random-Access Memory
RDD	Resilient Distributed Datasets
RDMA	Remote Direct Memory Access
RMM	Rack Management Module
RoCE	RDMA over Converged Ethernet
RSA	Rack Scale Architecture
RSD	Rack Scale Design
SaaS	Software as a Service
SAN	Storage Area Network
SATA	SerialAT Attachment
SCM	storage class memories
SCSI	Small Computer System Interface
SDCloud	Software-Defined Cloud
SDC	Software-Defined Computing
SDDC	Software-Defined DC
SDE	Software-Defined Environment
SDHI	Software-Defined "Hardware" Infrastructures
SDI	Software-Defined Infrastructures
SDN	Software-Defined Networking
SDON	software-defined optical network
SDS	Software-Defined Storage
SoC	System on Chip
SPMF	Scalable Platforms Management Forum
SR-IOV	Single-Root I/O Virtualization
SSD	Solid-State Drive
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TIP	Telecom Infra Project
TLB	translation lookaside buffer
ToR	Top of Rack
UPI	Ultra Path Interconnect
VM	Virtual Machine
VNF	virtual network function
XaaS	“X” as a Service