Composable Architecture for Rack Scale Big Data Computing

Chung-Sheng Li[1], Hubertus Franke[1], Colin Parris[2], Bulent Abali[1], Mukil Kesavan[3], Victor Chang[4]

1. *IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA*
2. *GE Global Research Center, One Research Circle, Niskayuna, NY 12309, USA*
3. *VMWare, 3401 Hillview Avenue, Palo Alto, CA 9430, USA*
4. *IBSS, Xi'an Jiaotong Liverpool University, Suzhou, China.*

csli@ieee.org, {frankh,abali}@us.ibm.com, colin.parris@ge.com, mukilk@gmail.com, ic.victor.chang@gmail.com

Keywords: Big data platforms, Composable system architecture, Disaggregated datacenter architecture, composable datacenter, software defined environments, software defined networking.

Abstract: The rapid growth of cloud computing, both in terms of the spectrum and volume of cloud workloads, necessitate re-visiting the traditional rack-mountable servers based datacenter design. Next generation datacenters need to offer enhanced support for: (i) fast changing system configuration requirements due to workload constraints, (ii) timely adoption of emerging hardware technologies, and (iii) maximal sharing of systems and subsystems in order to lower costs. Disaggregated datacenters, constructed as a collection of individual resources such as CPU, memory, disks etc., and composed into workload execution units on demand, are an interesting new trend that can address the above challenges. In this paper, we demonstrated the feasibility of composable systems through building a rack scale composable system prototype using PCIe switch. Through empirical approaches, we develop assessment of the opportunities and challenges for leveraging the composable architecture for rack scale cloud datacenters with a focus on big data and NoSQL workloads. In particular, we compare and contrast the programming models that can be used to access the composable resources, and developed the implications for the network and resource provisioning and management for rack scale architecture.

1 INTRODUCTION

Cloud computing is quickly becoming the fastest growing platform for deploying enterprise, social, mobile, and big data analytic workloads [1-3]. Recently, the need for increased agility and flexibility has accelerated the introduction of software defined environments (which include software defined networking, storage, and compute) where the control and management planes of these resources are decoupled from the data planes so that they are no longer vertically integrated as in traditional compute, storage or switch systems and can be deployed anywhere within a datacenter [4].

The emerging datacenter scale computing, especially when deploying big data applications with large volume (in petabytes or exabytes), high velocity (less than hundreds of microsecond latency), wide variety of modalities (structure, semi-structured, and non-structured data) involving NoSQL, MapReduce, Spark/Hadoop in a cloud environment are facing the following challenges: fast changing system configuration requirements due to highly dynamic workload constraints, varying innovation cycles of system hardware components, and the need for maximal sharing of systems and subsystems resources [5-7]. These challenges are further elaborated below.

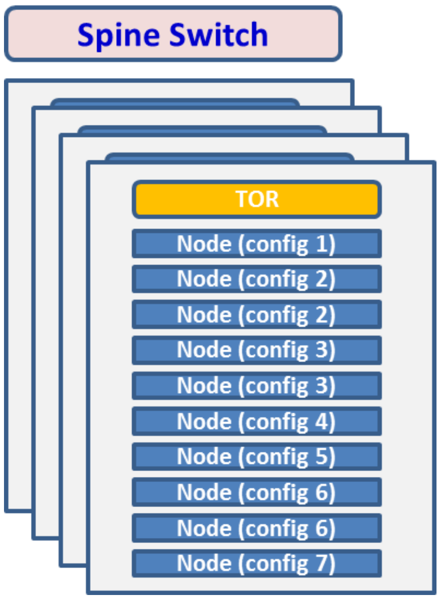

Figure 1 Traditional datacenter with servers and storage interconnected by datacenter networks.

Systems in a cloud computing environment often have to be configured differently in response to different workload requirements. A traditional datacenter, as shown in Fig. 1, includes servers and storage interconnected by datacenter networks. Nodes in a rack are interconnected by a top-of-rack (TOR) switch, corresponding to the leaf switch in a spine-leaf model. TORs are then interconnected by the Spine switches. Each of the nodes in a rack may have different CPU, memory, I/O and accelerator configurations. Several configuration choices exist in supporting different workload resource requirements optimally in terms of performance and costs. A typical server system configured with only CPU and memory while keeping the storage subsystem (which also includes the storage controller and storage cache) remote is likely to be applicable to workloads which do not require large I/O bandwidth and will only need to use the storage occasionally. This configuration is usually inexpensive and versatile. However, its large variation in sustainable bandwidth and latency for accessing data (through bandwidth limited packet switches) make it unlikely to perform well for most of the big data workloads when large I/O bandwidth or small latency for accessing data becomes pertinent. Alternatively, the server can be configured with large amount of local memory, SSD, and storage. Repeating this configuration for a substantial portion of the datacenter, however, is likely to become very expensive. Furthermore, resource fragmentation arises for CPU, memory, or I/O intensive big data workloads as these workloads often consume one or more dimensions of the resources in its entirety while left other dimensions underutilized (as shown in Fig. 2, where there exists unused memory for CPU intensive workloads (Fig. 2(b)) or unused CPU for memory intensive workloads (Fig. 2(c)). In summary, no single system configuration is likely to offer both performance and cost advantages across a wide spectrum of big data workload.

Traditional systems also impose identical lifecycle for every hardware component inside the system. As a result, all of the components within a system (whether it is a server, storage, or switches) are replaced or upgraded at the same time. The "synchronous" nature of replacing the whole system at the same time prevents earlier adoption of newer technology at the component level, whether it is memory, SSD, GPU, or FPGA. The average replacement cycle of CPUs is approximately 3-4 years, HDDs and fans are around 5 years, battery backup (i.e. UPS), RAM, and power supply are around 6 years. Other components in a data center typically have a lifetime of 10 years. A traditional system with CPU, memory, GPU, power supply, fan, RAM, HDD, SSD likely has the same lifecycle for everything within the

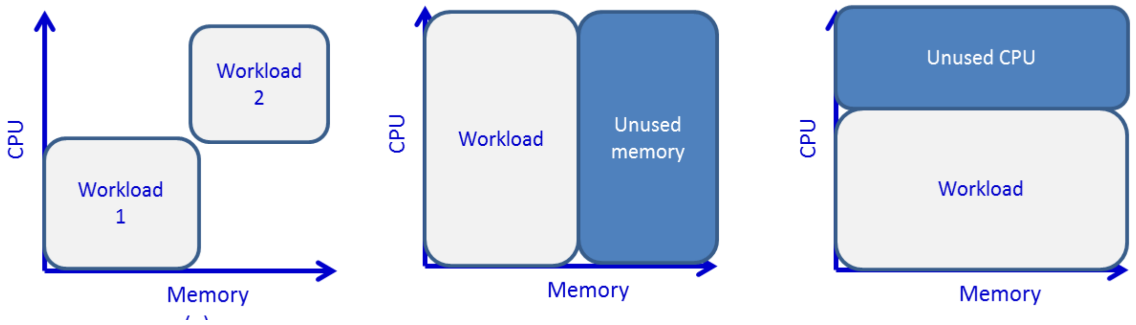

Figure 2: Fitting workloads to nodes in a cloud environment. (a) Typical workloads where CPU and memory requirements can be easily fit into a system. (b) CPU intensive workload with unused memory capacity. (c) Memory intensive workloads.

system as replacing these components individually will be uneconomical.

System resources (memory, storage, and accelerators) in traditional systems configured for high throughput or low latency usually cannot be shared across the data center, as these resources are only accessible within the "systems" where they are located. Using financial industry as an example, they are often required to handle large number of Online Transaction Processing (OLTP) during day time while conducting Online Analytical Processing (OLAP) and business compliance related computation during night time (often referred to as batch window) [8]. OLTP has very stringent throughput, I/O, and resiliency requirements. In contrast, OLAP and compliance workloads may be computationally and memory intensive. As a result, resource utilization could be potentially low if systems are statically configured for individual OLTP and OLAP workloads. Those resources (such as storage) accessible remotely over datacenter networks allow better utilization but the performance in terms of throughput and latency are usually poor, due to a prolonged execution time and constrained quality of service (QoS).

Disaggregated datacenter, constructed as a collection of individual resources such as CPU, memory, HDDs etc., and composed into workload execution units on demand, is an interesting new trend that satisfies several of the above requirements [9]. In this paper, we demonstrated the feasibility of composable systems through building a rack scale composable system prototype using PCIe switch. Through empirical approaches, we develop assessment of the opportunities and challenges for leveraging the composable architecture for rack scale cloud datacenters with a focus on big data and NoSQL workloads. We compare and contrast the programming models that can be used to access these composable resources. We also develop the implications and requirements for network and resource provisioning and management. Based on this qualitative assessment and early experimental results, we conclude that a composable rack scale architecture with appropriate programming models and resource provisioning is likely to achieve improved datacenter operating efficiency. This architecture is particularly suitable for heterogeneous and fast evolving workload environments as these environments often have dynamic resource requirements and can benefit from the improved elasticity of the physical resource pooling offered by the composable rack scale architecture.

The rest of the paper is organized as follows: Section 2 describes the architecture of composable systems for a refactored datacenter. Related work in this area is reviewed in Seciton 3. The software stack for such composable systems is described in Section 4. The network considerations for such composable systems are described in Section 5. A rack scale composable prototype system based on PCIe switch is described in Section 6. We describe the rack scale composable memory in Section 7. Section 8 describes the methodology for distributed resource scheduling. Empirical results from various big data workloads on such systems are reported and discussed in Section 9. Discussions of the implications are summarized in Section 10.

2 COMPOSABLE SYSTEM ARCHITECTURE

Composable datacenter architecture, which refactors datacenter into physical resource pools (in terms of compute, memory, I/O, and networking), offers the potential advantage of enabling continuous peak workload performance while minimizing resource fragmentation for fast evolving heterogeneous workloads. Figure 3 shows rack scale composability, which leverages the fast progress of the networking capabilities, software defined environments, and the increasing demand for high utilization of computing resources in order to achieve maximal efficiency.

On the networking front, the emerging trend is to utilize a high throughput low latency network as the “backplane” of the system. Such a system can vary from rack, cluster of racks, PoDs, domains, availability zones, regions, and multiple datacenters. During the past 3 decades, the gap between the backplane technologies (as represented by PCIe) [10] and network technologies (as represented by Ethernet) is quickly shrinking. The bandwidth gap between PCIe gen 4 (~250 Gb/s) [10] and 100/400 GbE [11] will likely become even less significant. When the backplane speed is no longer much faster than the network speed, many interesting opportunities arise for refactoring systems and subsystems as these system components are no longer required to be in the same "box" in order to maintain high system throughput. As the network speeds become comparable to the backplane speeds, SSD and storage which are locally connected through a PCIe bus can now be connected through a high speed wider area network. This configuration allows maximal amount of sharing and flexibility to address the complete spectrum of potential workloads. The broad deployment of Software Defined Environments (SDE) within cloud datacenters is facilitating the disaggregation among the management planes, control planes, and data planes within servers, switches and storage [4].

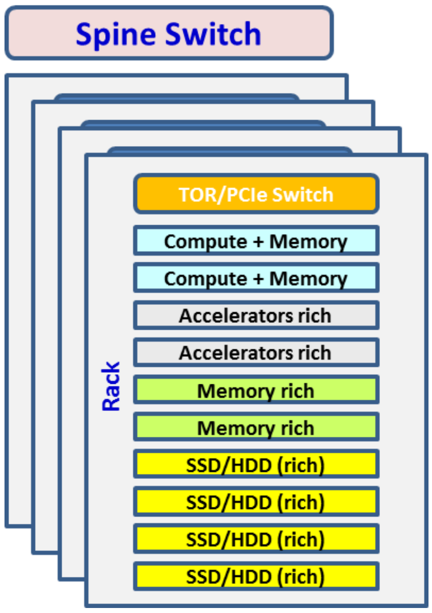

Figure 3: In rack scale architecture, each of the nodes within the rack is specialized into being rich in one type of resources (computing rich, accelerator rich, memory rich, or storage rich).

Systems and subsystems within a composable (or disaggregated) data center are refactored so that these subsystems can use the network "backplane" to communicate with each other as a single system. Composable system concept has already been successfully applied to the network, storage and server areas. In the networking area, physical switches, routing tables, controllers, operating systems, system management, and applications in traditional switching systems are vertically integrated within the same "box". Increasingly, the newer generation switches both logically and physically separate the data planes (hardware switches and routing tables) from the control planes (controller, switch OS, and switch applications) and management planes (system and network management). These switches allow the disaggregation of switching systems into these three subsystems where the control and management planes can reside anywhere within a data center, while the data planes serve as the traditional role for switching data. Similar to the networking area, storage systems are taking a similar path. Those monolithically integrated storage systems that include HDDs, controllers, caches (including SSDs), special function accelerators for compression and encryption are transitioning into logically and physically distinct data planes – often built from JBOD (just a bunch of drives), control planes (controllers, caches, SSDs) and management planes.

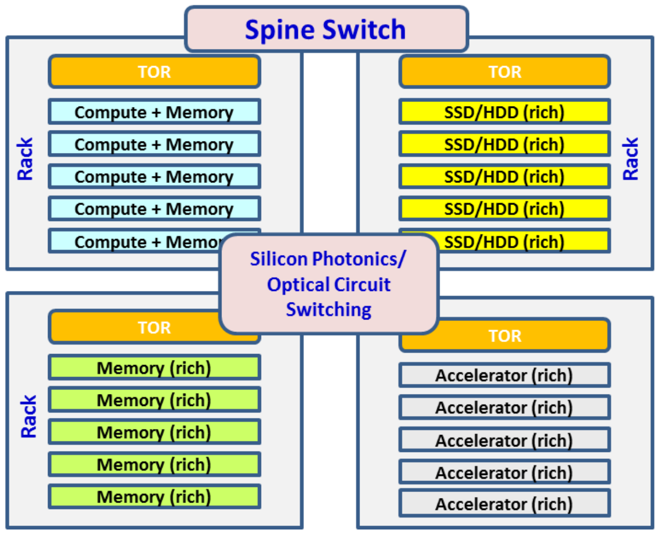

Figure 4: Disaggregation architecture applied at the PoD or datacenter level.

Figure 3 illustrates a composable architecture at the rack level. In this architecture, each of the nodes within the rack is specialized into being rich in one type of resources (computing rich, accelerator rich, memory rich, or storage rich). These nodes are interconnected by a low latency top-of-rack switch (and potentially a PCIe switch in addition to the TOR switch). In contrast to Fig. 1 where each of nodes within a rack may be configured differently (with different size or type of memory, accelerators, and local storage), there are far fewer node configurations in a rack scale architecture. The same concept can be extended to the PoD or datacenter level, as shown in Fig. 4, in which each rack consists of a specific type of nodes that have been specialized into computing rich, accelerator rich, memory rich, or storage rich nodes. In addition to the low latency TOR for providing connectivity at the rack level, low latency spine switch in a spine-leaf model or silicon photonics/optical circuit switches may be needed in order to maintain low latency between different racks.

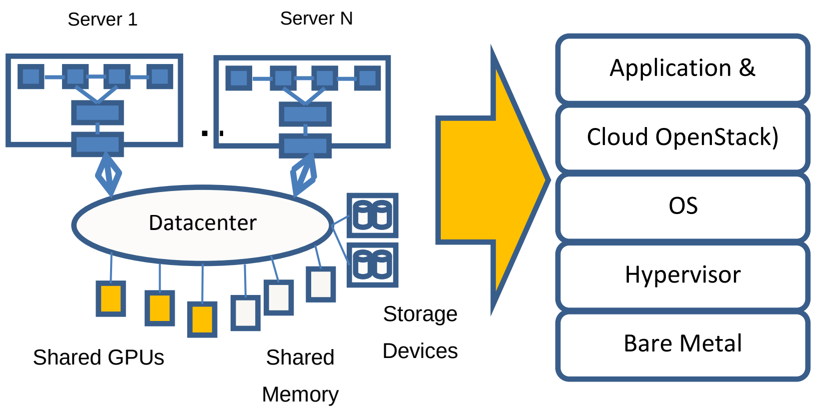

Figure 5: Software stack for accessing composable resources

The cost model for the *effective* cost of a system, C_{Total}, with N_{memory} of memory modules, N_{GPU} of GPU modules, N_{SSD} of SSD modules, and N_{HDD} of HDD modules, assuming the cost for each memory, GPU, SSD, and HDD module is C_{memory}, C_{GPU}, C_{SSD}, and C_{HDD}, respectively, and the utilization is U_{memory} , U_{GPU} , U_{GPU} , and U_{HDD} , respectively, can be defined in Eq. (1):

$$C_{Total} = N_{memory}\ C_{memory}/U_{memory} + N_{GPU}\ C_{GPU}/U_{GPU} + N_{SSD}\ C_{SSD}/U_{SSD} + N_{HDD}\ C_{HDD}/U_{HDD} \qquad (1)$$

The effective cost of a traditional system with utilization less than 50% for each type of resources is 33% higher than a composable system with 75% utilization for each type of resources.

3 RELATED WORK

High composability and resource pooling among CPUs, memory, and I/O resources is provided in a traditional Symmetric Multi-processing (SMP) with shared memory (scale up) architecture. The original logical resource partitioning concept – LPAR - was created during early 1970's as part of the IBM System 370 PR/SM (Processor Resource/System Manager) [12]. Subsequently, this concept was extended to DLPAR [13] to allow dynamic partitioning and reconfiguration of the physical resources without having to shut down the operating systems that runs in the LPAR. DLPAR enables CPU, memory, and I/O interfaces to be moved non-disruptively between LPARs within the same server. Virtual symmetric multiprocessing (VSMP) extended this concept in a scale out environment by mapping two or more virtual processors inside a single virtual machine or partition [14]. This makes it possible to assign multiple virtual processors to a virtual machine on any host having at least two logical processors.

Partially composable memory architecture was proposed by Lim et al [9, 15] in which each composable compute node retains a smaller size of memory while the rest of the memory is disaggregated and shared remotely. When a compute node requires more memory to perform a task, the hypervisor integrates (or “compose”) the local memory and the remote shared memory to form a flat memory address space for the task. During the run time, accesses to remote addresses result in a hypervisor trap and initiate the transfer of the entire page through RDMA (Remote Direct Memory Access) mechanism to the local memory. Their experimental results show an average of ten times performance benefit in a memory-constrained environment. A detailed study of the impacts of network bandwidth and latency of a composable datacenter for executing in-memory workloads such as GraphLab [16], MemcacheD [17] and Pig [18] was reported in [19]. When the remote memory is configured to contain 75% of the working set, it was found through simulation that the application level degradation was minimal (less than 10%) when network bandwidth is 40 Gb/s and the latency is less than $10\mu s$ [20].

There has been an ongoing effort to reconcile big data and big compute environments, such as the LLGrid at MIT Lincoln Lab [21]. Design and implementation of a lightweight composable operating system for composable processor sharing is reported in [22]. An in-memory approach for achieving significant performance improvement for big data and analytic applications was proposed for the traditional clustering and scale-out environment [23-25].

Large scale exploration of rack scale composable architecture has been demonstrated to produce substantial cost savings at Facebook for the newsfeed part of the Facebook infrastructure [26-27]. Server products based on a composable architecture have already appeared in the marketplace. These include the Cisco UCS M-Series Modular Server [28], AMD SeaMicro composable architecture [29-30], and Intel Rack Scale Architecture [31] as part of the Open Compute Project [32].

The focus of this paper, composable rack scale architecture, blends limited amount of resource pooling capabilities into a scale out architecture without requiring cache coherence (as compared to [13-14]) in this environment. The results and insights from earlier works for disaggregated systems as reported in [5, 15, 19] are largely obtained from simulation, and did not address some of the recent NoSQL workloads such as Giraph and Cassandra. In this paper, we reported prototyping effort for demonstrating rack scale composability using PCIe switch, and experimental results from running NoSQL and big data workloads such as Giraph, MemcacheD, and Cassandra.

4 SOFTWARE STACK

Composable datacenter resources can be accessed by application programming models through different means and methods. We consider and evaluate the pros and cons for three fundamental approaches, as shown in Fig. 5, including hardware based, hypervisor/operating system based, and middleware/application based.

The hardware based approach for accessing composable resources is transparent to applications and the OS/hypervisor. Hardware based composable memory presents a large and contiguous logical address space, which may be mapped into physical address space of multiple nodes, to the application. When the application accesses composable memory, the system management resolves the logical address of the request to the physical address within one of the compute nodes. In this case, the physical memory is byte addressable across the network and is entirely transparent to the applications. While such transparency is desirable, it forces a tight integration with the memory subsystem either at the physical level or the hypervisor level. At the physical level, the memory controller needs to be able to handle remote memory accesses. To avoid the impact of long memory access latencies, we expect that a large cache system is required. Composable GPU and FPGA can be accessed as an I/O device based on direct integration through PCIe over Ethernet. Similar to composable memory, the programming models remain unchanged once the composable resource is mapped to the I/O address space of the local compute node.

In the second approach, the access of composable resources can be exposed at the hypervisor, container, or operating system levels. New hypervisor level primitives - such as getMemory, getGPU, getFPGA, etc. - need to be defined to allow applications to explicitly request the provisioning and management of these resources in a manner similar to *malloc*. It is also possible to modify the paging mechanism within the hypervisor/operating systems so that the paging to HDD now goes through a new memory hierarchy including composable memory, SSD and HDD. In this case, the application does not need to be modified at all. Accessing remote Nvdia GPU through rCUDA [33] has been demonstrated, and has been shown to actually outperform locally connected GPU when there is appropriate network connectivity.

Details of resource composability and remoteness can also be directly exposed to applications and managed using application-level knowledge. Composable resources can be exposed via high-level APIs (e.g. Put/Get for memory). As an example, it is possible to define *GetMemory* in the form of *Memory as a Service* as one of the Openstack service. This potential Openstack *GetMemory* service will set up a channel between the host and the memory pool service through RDMA. Through this *GetMemory* service, the application can now explicitly control which part of its address space is deemed remote and therefore controls or is at least cognizant which memory and application objects will be placed remotely. In the case of *GPU as a service,* a new service primitive *GetGPU* can be defined to locate an available GPU from a GPU resource pool and host from the host resource pool. The system establishes the channel between the host and the GPU through RDMA/PCIe and exposes the GPU access to applications via a library or a virtual device. All of the experiments conducted in this paper are based on this approach, in which the composable resources are directly exposed to the applications.

5 NETWORK CONSIDERATIONS

One of the primary challenges for a composable datacenter architecture is the latency incurred over the interconnects and switches when accessing memory, SSD, GPU, and FPGA from remote resource pools. The latency sensitivity depends on the programming model used to expose composable resources in terms of direct hardware, hypervisor, or resource as a service. In order for the interconnect and switch technologies to be appropriate for accessing remote physical resource pools, the round trip access latency has to be insignificant compared to the inherent access latency of the resource so that the access of the resource can remain transparent to the applications. When the access latency of the remote resource pool become noticeable compared to the inherent access latency, there might be significant performance penalty unless thread level parallelism is exploited at the processor, hypervisor, OS, or application levels.

The most stringent requirement on the network arises when composable memory is mapped to the address space of the compute node and is accessed through the byte addressable approach. The total access latency across the network cannot be significantly larger than the typical access time of locally attached DRAM so that the execution of threads within a modern multi-core CPU can remain efficient. The bandwidth and latency for accessing locally attached memory through DMI/DDR3 interface today is 920 Gb/s and 75 ns, respectively. PCIe switch (Gen 3) can achieve latency on the order of 150 ns while low latency Top-of-Rack IP switch and Infiniband switch can achieve 800 ns latency or less. As a result, silicon photonics and optical circuit switches (OCS) are likely to be the only options to enable composable memory beyond a rack [34-36]. Large caches can reduce the impact of remote access. When the block sizes are aligned with the page sizes of the system, the remote memory can be managed as extension of the virtual memory system of the local hosts through the hypervisor and OS management. In this configuration, locally attached DRAM is used as a cache for the remote memory, which is managed in page-size blocks and can be moved via RDMA operations.

Disaggregating GPU and FPGA are much less demanding as each GPU and FPGA are likely to have its local memory, and will often engage in computations that last many microseconds or milliseconds. So the predominant communication mode between a compute node and composable GPU and FPGA resources is likely through bulk data transfer. It has been shown by [37] that adequate bandwidth such as those offered by RDMA at FDR data rate (56 Gb/s) already demonstrated superior performance than a locally connected GPU.

Network latency measurement is important since it can affect the performance in data center technologies including high performance computing, storage and data transfer between different sites, whereby the impact on network latency on the data center performance for Cloud and non-Cloud solutions is investigated in [38]. With the advancement in our proposed data technologies, regular measurement is not required since current SSD technologies have 100K IOPS and 100 us access latency. Consequently, the access latency for non-buffered SSD should be on the order of 10 us. This latency may be achievable using conventional Top-of-the-Rack (TOR) switch technologies if the communication is limited to within a rack. A flat network across a PoD or a datacenter with a two-tier spine-leaf model or a single tier spline model is required in order achieve less than 10 us latency if the communication between the local hosts and the composable SSD resource pools are across a PoD or a datacenter.

Table 1 summarizes the type of networks required for supporting composability from physical resource pools (memory, GPU/FPGA, SSD and HDD) at the Rack, PoD, and Datacenter levels. The entries in this table are derived from the considerations that the total round trip latency for accessing the remote physical resource pools has to be insignificant compared to the inherent access latency. The port-to-port latency for various interconnects and switch technologies are:

- Low latency TOR switch, such as those made by Arista (380-1000 ns) [39]
- Low latency spine-leaf switches, such as those made by Arista (2-10 us) [39]
- InfiniBand switch, such as those made by Mellanox (700 ns) [40]
- Optical circuit switch, such as those made by Calient (<30 ns) [41]
- PCIe switch, such as those made by H3 Platform (~150ns) [42]

The round trip propagation delay, assuming 5 ns/m, for rack, PoD, and datacenter are:

- Intra-rack: the average propagation distance is less than 3 m or 15 ns.
- Intra PoD: the average propagation distance is 50 m or 250 ns.
- Intra datacenter: the average propagation distance is 200 m or 1 us.

Consequently, rack level systems with composable GPU/FPGA, SSD, and HDD can be easily accommodated by low latency TOR switch, PCIe switch or InfiniBand switch. Low latency flat network based on spine-leaf switches become the primary option for PoD and datacenter level interconnect for composable resources.

	Memory (DMI/DDR3)	**GPU/FPGA (PCIe gen3)**	**SSD (SATA/SAS)**	**HDD (SATA/SAS)**
Latency	~75ns	5-10 us	~100 us	~25 ms
Throughput	~920 Gb/s	12 GB/s	~100K OPS	75-200 OPS
Rack (~3m, ~15ns)	Silicon photonics	TOR switch, PCIe switch, Infiniband switch	TOR switch, PCIe switch Infiniband switch	TOR switch, PCIe switch Infiniband switch
PoD (~50m, ~250ns)	Optical Circuit Swtich (OCS)	Flat network (Spine-Leaf), OCS	Flat network (Spine-Leaf), OCS	Flat network (Spine-Leaf), OCS
Datacenter (~200m, ~1us)	Optical Circuit Switch (OCS)	Flat network (Spine-Leaf), OCS	Flat network (Spine-Leaf), OCS	Flat network (Spine-Leaf), OCS

Table 1: Types of network for supporting composable systems at the rack, PoD and datacenter levels.

6 RACK SCALE COMPOSABLE I/O PROTOTYPE

Composable I/O is a special case of leveraging high throughput low latency network (often based on PCIe switch or Infiniband switch) to support physical resource pooling and reduce resource fragmentation at the rack level. PCIe fabrics do not scale beyond a few racks. However with the use of PCIe fabrics, resource pooling is simplified at the rack scale. Main advantage of PCIe fabrics over Ethernet, Fiber Channel, or Infiniband is that a PCIe fabric requires virtually no changes to the software stack, as a peripheral allocated from a PCIe-connected resource pool appear to be local to each server.

Most of the cloud datacenters use 2-socket rack servers in 1U (1.75-inch high) or 2U (3.5-inch high) physical form factors. 1U and 2U servers typically contain 1-2 and 4-6 PCIe slots, respectively. When high compute density is required, 1U servers may be used at the expense of limited number of PCIe slots. As a result, these 1U servers are specialized according to the functions associated with the PCIe slots such as SSD or GPU-specialized servers. When PCIe rich servers are required, a 2U server must be used at the expense of reduced compute density and often vacant PCIe slots. Either scenario leads to inefficiencies in the datacenter. This is due to mismatches between available system resources configured in 1U and 2U and the dynamically changing resource requirements for big data workloads. The composable I/O architecture addresses the I/O aspect of this problem by physically decoupling the I/O peripherals from individual servers. In one realization of this composable I/O architecture, PCIe peripherals are aggregated in a common I/O drawer in the rack (as shown in Fig. 3). The challenges for realizing such architecture include achieving multiplexing, scalability, sharing, and reliability:

- **Multiplexed I/O** is the ability to dynamically attach and detach any PCIe peripheral to any server. Multiplexed I/O is essential in "bare-metal" clouds in which customers rent and compose server resources on-demand, such as the type and number of PCIe peripherals. Multiplexed I/O should also allow logical hot plugability of a PCIe card to a running server.
- **Scalable I/O** is the ability to attach large number of PCIe cards to a single server regardless of its form factor, hence achieving better compute density and resource utilization at the same time.
- **Shared I/O** is the ability to share a single PCIe card among multiple hosts, therefore reducing the infrastructure cost.
- **Reliability** refers to the highly available and fault tolerant I/O fabrics.

Addressing all these challenges transparently without impacting the software stack is desirable. Supporting Multiplexed I/O and Scalable I/O requires no changes in the software stack. PCIe peripherals simply appear to be local to the server through PCIe switch fabric. A management layer for the composable I/O fabrics may be required. Supporting shared I/O requires MRIOV capable adapters or some adaptation of SRIOV as detailed below.

Recent advances in PCIe switching technologies enable inexpensive implementation of the composable I/O architecture. PCIe bus in a traditional server is located on its motherboard. A "southbridge" chip served as a gateway from processor and memory to the PCIe slots on the motherboard. Newer PCIe switch chips (e.g., from the PLX and IDT) allow interconnecting multiple hosts and PCIe peripheral devices. These switch chips (e.g. PLX PEX9797) can provide up to 96 one-bit wide (x1) PCIe lanes that can be combined to create a crossbar switch of port sizes at x1, x4, x8, and x16. Multiple switch chips can be cascaded to provide even larger PCIe fabric. Virtual PCIe networks within each fabric may be software defined, giving each root the ability to host the private ownership of the downstream PCIe devices. Additionally, host-to-host communication is made possible by creating remote memory access tunnels and on-chip DMA engines with NIC-like functionality, albeit in a non-standard fashion. PCI-SIG (the standard group responsible for defining PCI) has defined the Multi-Root I/O Virtualization (MR-IOV) extension of the PCIe spec for multiple hosts sharing a single PCIe adapter. Industry adoption of MR-IOV has yet to happen since its inception almost a decade ago. On the other hand, Single-Root I/O Virtualization (SR-IOV) [43] adapters have been more broadly embraced within the industry by cloud service providers such as AWS and has been demonstrated to achieve up to 95% of the bare metal adapter card [44-45]. SR-IOV adapters were originally intended for hypervisor-based hosts. The adapter presents multiple virtual end points to the virtual machines on a single host, with the hypervisor responsible for managing the physical device. It has been demonstrated that MR-IOV adapter can be emulated by the PEX9x switch chips to provide connection from multiple physical hosts to virtual interfaces on a single SR-IOV adapter [43]. With host-to-host communication and I/O sharing capability, it is conceivable that cloud service providers will leverage PCIe switch and SR-IOV NIC combination to eliminate or reduce the port counts of the top-of-rack (TOR) switches. To sum up, we expect rack scale composable I/O solutions (for example, a chassis of PCIe cards that multiple hosts attach to) to become more broadly adopted.

A number of recent works demonstrated the benefit of applying composable I/O functionality in a cloud environment. A novel use of PCIe switching fabric in the Pelican Cold Storage prototype [46] demonstrated highly scalable I/O fabrics. Pelican is a rack-scale hard disk based storage prototype for "cold" data that are only infrequently accessed. In the Pelican prototype, only 96 drives out of 1152 (~8.3%) are powered on at any given time to minimize the rack power and cooling requirements. The novel use of PCIe switches come in to play when interconnecting the 1152 drives to the two hosts without using SAN storage controllers.

Single host with multiple GPU systems have been experimented in the high performance computing community [47-48]. A cluster of nodes with 8 GPUs per host can be created using PCIe switches [49], as demonstrated by the Facebook Big Sur system [50]. Workloads involving GPU computations are often bandwidth intensive and therefore the single rooted PCIe tree can easily become a bottleneck. Consequently, peer-to-peer copy is used by GPUs for directly exchanging data with no staging thru the host memory.

The IBM zEnterprise 196 I/O subsystem demonstrates the reliability aspect of the composable I/O concept [51]. The I/O subsystem is contained in a 32-slot I/O drawer (each slot is PCIe Gen2 x8). The I/O drawer is divided in to 4 I/O domains of 8 PCIe slots each. Each domain contains a 96-lane PCIe switch with x8 downlinks and one x16 uplink connected to one of the four SMP processor cards. An x16 failover link interconnects the two PCIe switches in two different I/O domains. When an SMP processor card becomes offline for any reason (maintenance or faults), the x16 uplink goes down. The x16 failover link provides access to the PCIe peripherals through another processor card.

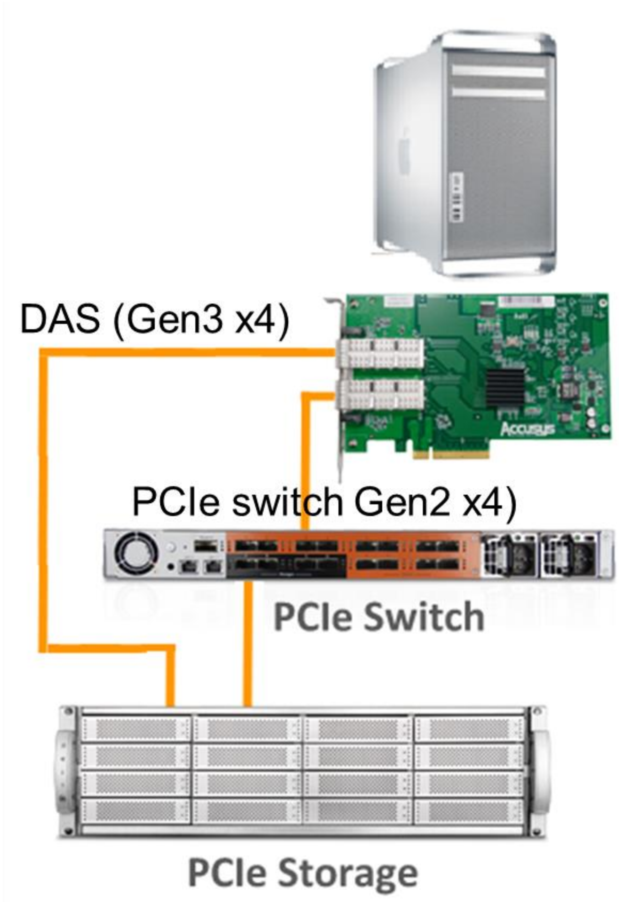

Figure 6: PCIe shared storage experimental setup

	DAS (no PCIe switch)		PCIe switch	
	BW MB/s	IOPS	BW MB/s	IOPS
Sequential 4KB read (iodepth=16)	1684	431,182	1193	305,648
Sequential 4KB read (iodepth=64)	1624	415,699	1344	344,029

Table 2: PCIe shared storage experimental results

The prototype rack scale composable system based on PCIe switch developed for this paper is shown in Fig. 6. It consisted of an H3 RAID array with 12 SATA drives with a single PCIe Gen3 x4 port, a PCIe switch (SW16) with 16 Gen2 x4 ports, and Host Bus Adapter (HBA) plugged in to an IBM 3650M4 server. The HBA had two ports, one labeled SAN for connecting thru SW16 and the other one labeled DAS for an optional direct connection to the RAID array. The SW16 switch contains several stages of PCIe switch chips and adds 450 nanoseconds of latency. Note that SW16 ports were Gen2, therefore both the RAID box and the HBA would step down to Gen2 speed when connected to SW16, whereas the direct DAS connection ran at Gen3 speed.

The sequential 4KB read experiment is conducted on the prototype to determine the impact on the I/O latency and throughput due to the PCIe switch introduced in the composable rack scale system. Only sequential read is conducted as random reads on HDDs often incur additional latency due to access time (often exceeds 25 ms) resulting from the seek time of the disk head and rotation delay of the platter. Benchmark had four software threads pinned to specific CPUs and initiated read operations with I/O depths of 16 and 64, indicating the number of asynchronous I/Os in progress at once.

Table 2 shows that the bandwidth achieved without and with PCIe switch setups were 1684 MB/s and 1193 MB/s, respectively. The IOPS rates without and with PCIe switch were 431K and 305K respectively. These measurements indicate that the Gen2 PCIe switch introduces 29% reduction on both the throughput and the IOPS rate. Subsequently, the four software threads' IO depth was increased from 16 to 64 to overlapped I/O requests to compensate for the PCIe switch latency. It was observed that the bandwidth increased from 1193 MB/s to 1344 MB/s and the IOPS rate increased from 305K to 344K. However, this still represents 17% penalty on the bandwidth and IOPS rate when compared to the configuration without the Gen2 PCIe switch. Again, most of the penalty can be explained by the throughput differences between Gen3 vs. Gen2 of PCIe operations, which are 8 GT/s and 5 GT/s, respectively, for the base standard.

PCIe switch fabric is expected to continue improving during the coming years. The next step of the PCIe switch based rack scale composable system will likely to be (1) based on Gen3 PCIe switch fabric (2) supporting larger number of ports (>96 ports) (3) continued evolution of the multi-root support.

7 RACK SCALE COMPOSABLE MEMORY

Rack scale composable memory intends to reduce the memory fragmentation problem in the datacenter. Various workload analysis demonstrated that 30~50% (and sometimes up to 90%) of server memory capacity goes unused [52-55]. Many of these scenarios indicated that CPU capacity is exhausted before memory capacity is reached, therefore leaving a fraction of the memory unused. It is hypothesized that when the memory is placed in shared pools, higher efficiencies may be achieved by composing servers dynamically through carving out the necessary amounts of memory from these shared pools [9, 15-19]. However these isolated analyses overlooked the performance and cost issues of memory disaggregation. Modern microprocessors and DRAMs often have strong affinity to each other. Pooling the memory in a centralized location, either at a rack scale or at datacenter scale, will impact memory performance and cost. First, bandwidth of state-of-the-art memory channels is couple orders of magnitude higher than that of the existing network fabrics. Signal integrity and high bandwidth requirements of long links will require optical interconnects but at a much increased cost [56]. Second, disaggregation will increase the memory latency due to increased physical distance and additional latency introduced by the switching.

The latency of reading one cache block, typically 64 to 128 bytes, from memory to the processor cache takes approximately 75 nanoseconds. Additional latency is incurred in distributed applications for message passing when messages are sent over Ethernet or InfiniBand fabrics between nodes in a cluster. The impact on the performance of these applications due to longer network latency is lessened by using large packet sizes and asynchronous messaging. However, these approaches are not practical at the processor instruction level. As applications access remote memory explicitly (e.g. using RDMA), they most likely utilize a distributed memory cluster environment supporting message passing as opposed to a pooled memory environment supporting synchronous load/stores instructions from processors.

The processor performance as a function of memory latency is simulated using a cycle accurate processor simulator. The simulated instruction traces of several benchmarks using different memory latencies indicated a linear relationship between performance and latency. Two hypothetical processors P1 and P2 with 12 and 16 core, respectively, were simulated. Each core has an L1 size of 64KB, L2 size of 512KB, and L3 size of 8MB. Both P1 and P2 use out-of-order execution. P1 can issue 10 instructions per cycle to 16 functional units in each core. Each core can have up to 16 outstanding cache misses. Both processors implement simultaneous multithreading / hyperthreading (SMT) found in x86 and POWER processors [57]. In the SMT mode, each core supports 2, 4 or 8 logical processors that share its functional units and the execution pipeline, therefore increasing the total core throughput by a factor of 2 to 3. The SPEC CPU2006 Integer and Floating Point suite of benchmark [58] traces were simulated on P2. Four commercial benchmarks OLTP, ERP, TRADE, and SALES were simulated on P1. OLTP is an online transaction processing benchmark that measures the rate of queries/transactions performed on a database. TRADE is a Java based stock trading application. ERP is an enterprise resource planning application. SALES is a customer order processing and distribution application.

The workload's sensitivity to memory latency, Memory Fraction of Performance (***MFP***), is defined in Eq. (2)

$$\boldsymbol{MFP} = \Delta\boldsymbol{ET} / \Delta\boldsymbol{ML} \tag{2}$$

where $\Delta\boldsymbol{ET}$ is the relative increase in benchmark execution time while $\Delta\boldsymbol{ML}$ the relative increase in the memory latency. Both $\Delta\boldsymbol{ET}$ and $\Delta\boldsymbol{ML}$ are relative to its baseline value. In essence, MFP is the fraction of execution time attributable to the memory latency. For example, an MFP of 40% indicates that the memory latency is responsible for 40% of the execution time. Execution time would increase by 40% if the memory latency doubled from a base of 75ns to 150ns. A workload with a small MFP is insensitive to the memory latency, as its working set fits in to the processor's on-chip caches.

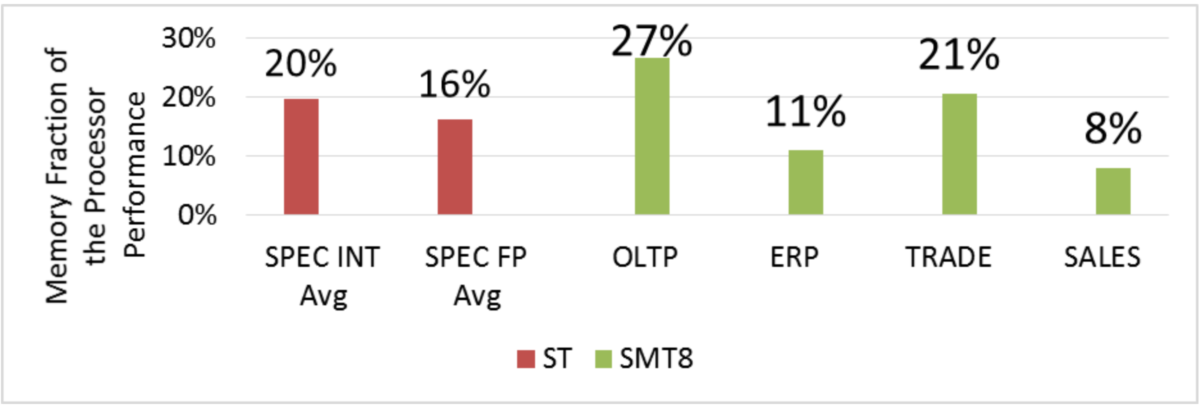

Figure 7: The fraction of the execution time due to memory latency

Fig. 7 summarizes the memory latency sensitivity for commercial and SPEC benchmarks. Average MFP for the INT and FP suites are 20% and 16%, respectively, on single threaded (ST) cores. Individual benchmarks (not detailed here) had an MFP as high as 59%, which demonstrated that composable memory architecture at the PoD or datacenter level is unlikely to be suitable for these types of applications where the application performance is sensitive to the memory latency. In a comosable memory architecture when the processor and memory are more than a few racks apart (with a distance up to 10 meters will introduce a round trip delay of 2 x 10 meters x 5ns/m = 100ns), resulting in more than doubling the base latency.

A noteworthy observation is that MFP decreases with increasing SMT levels for most of the benchmarks. In the case of SPEC INT, the average MFP is reduced from 20% to 8% when the simultaneous thread level is increased from single thread to 8 concurrent threads. In other words, workloads are more tolerant to increasing memory latencies on an SMT processor. This is because the threads of a core may have to wait for each other while accessing the shared functional units and pipeline stages, and hides some portion of the memory latency. Results suggest that disaggregated memory systems and other high latency memory systems will benefit from high SMT parallelism.

In a disaggregated memory system, the memory latency may increase because of the additional delay contributed by the signal propagation, switches (queueing/buffering), E/O & O/E (electrical-to-optical and optical-to-electrical conversion), serialization/deserialization, line coding/decoding, and protocol conversion. Propagation of light in optical fiber will add ~5 nanoseconds (ns) to the latency per meter. A memory chassis in a single rack may add as much as 6 meters roundtrip (30 ns or 40%) to the base latency of 75 ns. Memory racks serving an entire data center will add ~500 ns (or ~100m) to the latency. A memory "switch" required for a pooled memory may add 75 to 150 ns. Therefore depending on the scale of disaggregation, we estimate that the total memory latency will be at minimum 150 ns, resulting in at least an increase of 40% of the execution time for the SPEC INT suite. Thus, the increased resource utilization efficiency due to disaggregation may be partially or entirely offset by the increase in processor costs when the additional latency cannot be hidden [56].

Another design approach is managing the local/remote (pooled) memory hierarchy using software, namely the virtual memory subsystem of the OS or the hypervisor. Referencing to pages not present in the local memory is trapped by VMM and those missing pages will be retrieved from the pooled memory in to the local memory by using, for example, RDMA. Alternatively, the application software can be modified to provide the management of the local vs. remote memory explicitly. Software overhead of this approach can be considerable, and often exceeds 1 microseconds per page. An increase of 1 to 10 microsecond memory latency will increase the execution time by 266% and 2666%, respectively, for the SPEC INT suite according to Eq. (1). Note also that managing the two level local/remote memory at the page granularity significantly increases the bandwidth consumption of the memory and the fabric, since not all data in a given page are touched by the processor [59]. In Section 9, we present experimental results of this software managed remote memory approach.

In summary, retaining sufficient local DRAM serving as the cache for the pooled memory as opposed to full disaggregation of memory resources and retain no local memory for the CPU is always recommended to minimize the performance impact due to latency incurred from accessing remote memory, regardless whether the access is managed by hardware, OS, or applications. Higher SMT levels [57] and/or explicit management by applications that maximize thread level parallelism are also essential to further minimize the performance impact.

8 DISTRIBUTED RESOURCE PROVISIONING

In a composable datacenter with physical resource pooling, it is essential that the physical resources are requested and provisioned with minimum latency so that the use of remote resources will not create a serious performance bottleneck. A traditional scheduling environment provisions resources based on the maximal anticipated resource requirement for the duration of a workload. In this section, an approach based on distributed scheduling with global shared state in conjunction with predictive resource provisioning is proposed. Resource provisioning and scheduling can be carried out through centralized, hierarchical, or fully distributed approaches. The centralized approach is likely to achieve the optimal resource utilization, but may result in a single point of failure and a severe performance bottleneck. The hierarchical approach, such as the one used in Mesos [60], allows flexible addition of heterogeneous schedulers for different classes of workloads to a centralized scheduler. The centralized scheduler allocates chunks of resources to the workload specific scheduler, which in turn allocates resources to individual tasks. However, this approach often results in sub-optimal utilization. A fully distributed approach with global shared state, such as the Google Omega [61] project, utilizes an optimistic approach for resource scheduling. This approach is likely to perform better as compared to other approaches.

The mechanism for scheduling and provisioning resources from composable physical resource pools starts with the requesting node establishing the type and amount of resource required. As discussed in the previous section, the amount of resource required can be established explicitly by the workload or implicitly as the current requesting node runs out of resource locally. Once the request is received, the resource provisioning engine will identify one or more of the resource pools with available resources, potentially based on the global shared state, for provisioning resources. It will then communicate with the resource manager of the corresponding resource pool to reserve the actual resource. The resource manager for each resource pool commits the resource to the incoming request and resolves the potential conflicts if multiple requests for the same resource occur simultaneously. Once the resource is reserved, the communication between the requesting node and the resource can then commence.

Due to the low latency requirement for provisioning physical resources in a composable datacenter, it is likely that the resources will need to be provisioned and reserved before the actual needs from the workload arise rather than on demand. This may require the resource scheduler to monitor the history of the resource usage so that an accurate workload dependent projection of the resource usage can always be maintained. The impact to the resource utilization due to advanced reservation can be minimized by (1) maintaining a distributed global resource state, and (2) utilizing opportunistic based distributed reservation scheme such as the methodology reported in [62] to minimize scheduling latency and hence the required advanced reservation.

The primary challenges in developing a workload forecasting mechanism include [63-66, 69-71]: (1) potential overheads related to change of provisioned resources as it will take time to properly set up resources before they can be used by the workload, (2) ability to accurately predict future workload behavior, and (3) ability to compute the right amount of resources required for the expected increase or decrease in workload [62]. The general framework of such a scheduling mechanism can be represented by the pseudocode below:

```
Initialize Observation window
Initialize Prediction window
While (Workload is in progress) {
  Generate predicted memory requirement for the next Prediction Window from the current Observation
  Window;
  Provision memory based on the memory requirement;
  Generate predicted accelerator requirement for the next Prediction Window from the current
  Observation Window;
  Provision accelerator based on the accelerator requirement;
Generate predicted IO requirement for the next Prediction Window from the current Observation Window;
Provision IO based on the IO requirement;
}
```

In this mechanism, an ***observation window*** of length w is set up for the workload to collect the behavior pattern in terms of resource consumption of the workload. A prediction function is defined to predict the peak usage amount of the specific resource type (memory, accelerators, and IO) during the prediction window. The simplest prediction mechanism can be based on autoregressive moving average (ARMA) [62] based on the workload behavior pattern collected during the observation window:

$$\begin{aligned}
&\mathbf{M}(t+1) = a + a_0\,\mathbf{M}(t) + a_1\,\mathbf{M}(t-1) + \ldots. + a_{w-1}\,\mathbf{M}(t-w+1)\\
&\mathbf{A}(t+1) = b + b_0\,\mathbf{A}(t) + b_1\,\mathbf{A}(t-1) + \ldots. + b_{w-1}\,\mathbf{A}(t-w+1)\\
&\mathbf{IO}(t+1) = c + c_0\,\mathbf{IO}(t) + c_1\,\mathbf{IO}(t-1) + \ldots. + c_{w-1}\,\mathbf{IO}(t-w+1)
\end{aligned} \tag{3}$$

where a_i, b_i, and c_i ($i = 0,\ldots, w-1$) are the ARMA coefficients, and $\mathbf{M}(t+1)$, $\mathbf{A}(t+1)$, and $\mathbf{IO}(t+1)$ are the predicted memory, accelerator, and IO requirements, respectively. More sophisticated resource estimation models including those based on machine learning techniques such as neural networks have been developed for workloads ranging from transaction oriented (i.e. OLTP) to data intensive computations [63-66]. Based on the predicted resource requirements from the observation window, the execution environment can then provision the resources for the next prediction window:

- *GetMemory (Predicted_Memory_Requirement)*
- *GetAccelerator (Predicted_Accelerator_Requirement)*
- *GetIO (Predicted_IO_Requirement)*

Provisioning resources based on shorter term needs of the workload enable more aggressive resource sharing among workloads. This provisioning mechanism becomes similar to the traditional provisioning mechanism when the prediction window approaches the entire duration of the workload execution.

9 EXPERIMENTAL RESULTS

In this section, we describe experiments that demonstrate the workload behavior when a cloud centric big data or NoSQL application such as MemcacheD, Giraph, and Cassandra is deployed in a composable system environment where remote memory or storage is exposed at the middleware/application level through simplified API.

a. MemcacheD workload

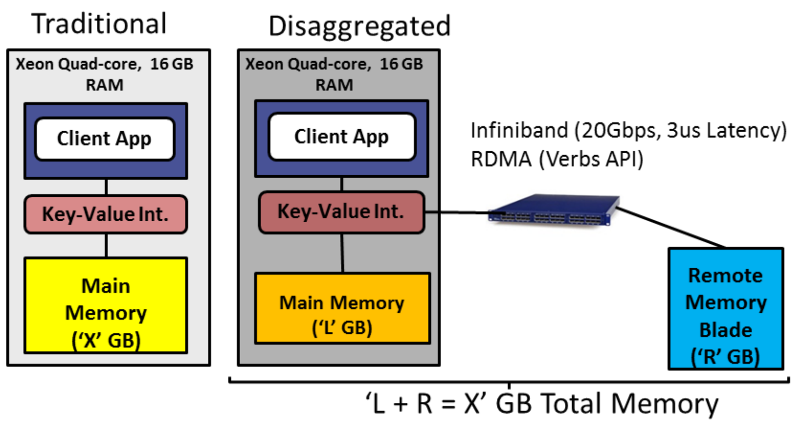
Traditional
Disaggregated
Xeon Quad-core, 16 GB RAM
Client App
Key-Value Int.
Main Memory ('X' GB)
Xeon Quad-core, 16 GB RAM
Client App
Key-Value Int.
Main Memory ('L' GB)
Infiniband (20Gbps, 3us Latency)
RDMA (Verbs API)
Remote Memory Blade ('R' GB)
'L + R = X' GB Total Memory

Figure 8: Experimental setup for performance measurement in a composable environment for MemcacheD.

In this environment, part of client application data is in local DRAM, while the rest is located in the memory of a remote node accessed through an RDMA capable fabric via the Verbs API [67]. The composable infrastructure, as shown in Fig. 8, is entirely transparent to the MemcacheD client. The server side is modified so that the data accessed via key-value interface will be automatically retrieved from either local or remote memory.

The experiment is as follows: A small program on a remote machine allocates a specified amount of memory and registers the allocation with the InfiniBand HCA. MemcacheD handshakes with the remote server and obtains the pertinent information such as remote buffer address and access_key. After an initial handshake, it can now perform RDMA reads and writes directly to the remote buffer. The remote buffer is treated as a "victim cache" and is maintained as an append-only log. When MemcacheD runs out of local memory, instead of evicting a key/value pair in the local memory, it now does an RDMA write to the remote memory. When looking up a particular key, it first checks with the local memory (via a hash table). If the key does not exist locally, MemcacheD checks the remote memory via a locally maintained hash table. If key/value is in the remote memory, it reads in this value through RDMA to a temporary local buffer and sends it to the client. A particular key/value is always either in local memory or remote memory and can never reside in both locations.

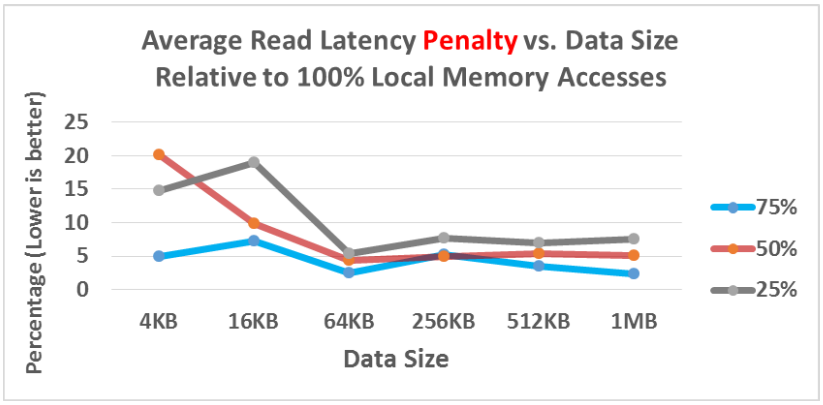

Figure 9: Average read latency penalty vs. data size with respect to 100% local access when the local portion of data varies from 75% to 25%

The experiments consist of 100,000 operations (95% reads, 5% updates) with uniform random accesses (i.e. no notion of working set as this represents the most challenging situation) running in a single thread.

As shown in Fig. 9, higher percentage of local data always introduces fewer penalties. However, the difference begins to diminish among different ratio of local vs. remote data when the data block size is larger than 64 KB, as larger block size reduces the overhead in the data transfer.

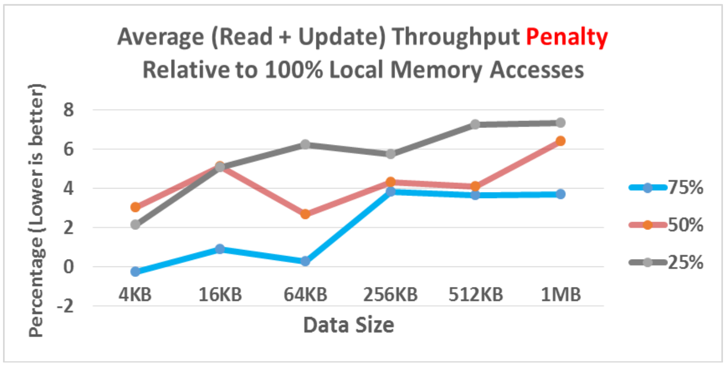

Figure 10: Average read/update throughput penalty vs. data size with respect to 100% local access when the local portion of data varies from 75% to 25%.

The second set of experiments consist of 100,000 read and update operations (95% reads, 5% updates) with uniform random accesses (i.e. no notion of working set as this represents the most challenging situation) evenly split among 10 threads.

As shown in Fig. 10, the throughput penalty is nearly nonexistent when 75% of the access is local and the data size is 4KB. The penalty increases to 2% when only 25% of the access is local. As the data sizes increase, the transfer time of the entire page between the local and the remote node increases, resulting in higher penalty at 4% and 6%, respectively, for 75% and 25% local access.

We can conclude from these experiments that negligible latency and throughput penalty are incurred for the read/update operations if these operations are 75% local and the data size is 64 KB. Smaller data size results in larger latency penalty while larger data size results in larger throughput penalty when the ratio of nonlocal operations is increased to 50% and 75%.

b. Giraph workload

The second experiment focuses on the popular graph analytics platform Giraph, which enables implementation of distributed vertex-centric graph algorithms. The goal is to quantify the memory usage of a popular graph algorithm in order to identify opportunities for running it in a composable memory environment. In this particular case, a 50-node virtual compute cluster is populated with a randomly generated graph with 100 million vertices. The graph is partitioned into 50^2 partitions and are distributed evenly across the computing nodes. The TopKPagerank algorithm [68] is then run on the entire graph for a fixed number of supersteps. As the computation progresses, messages need to be exchanged to traverse the graph as the computation crosses node boundaries. Dependent on the connectivity of the graph, the variance in the message creation can result in substantially different memory consumptions per node. When the available memory is constrained, Giraph will swap the entire partitions and the messages associated with the vertex to disk using LRU. The memory utilization across the nodes is monitored as computations progresses. While CPU utilization is very uniform across all nodes and across the execution of the program, memory utilization varies considerably, which is shown as a heatmap in Fig. 11.

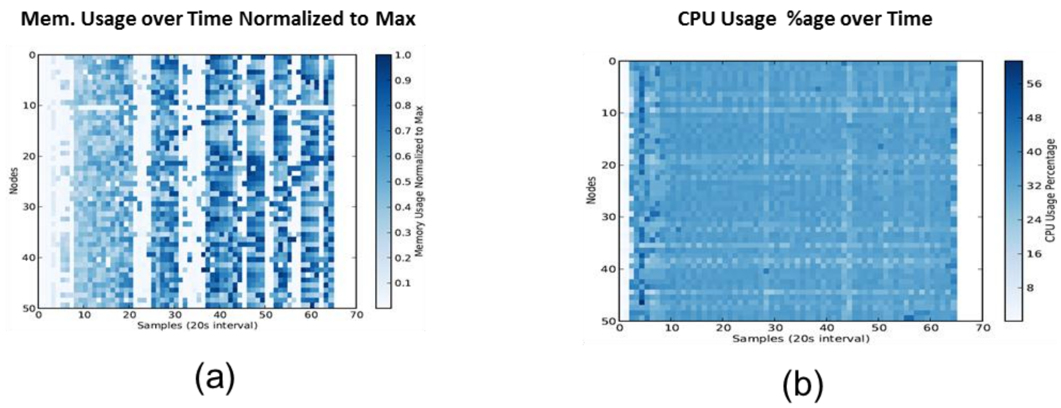

Figure 11: (a) Memory and (b) CPU Consumption of Distributed Giraph TopKPagerank application over time.

Analysis of this data reveals that per node memory usage between peak and average has a 2.78:1 ratio, where the aggregate memory usage has a 1.68:1 ratio. The memory per node is then reduced by a factor of 3 to explore the impact of memory pressure, while the average per node memory is maintained. This increases the overall runtime of the experiment by a factor of 13.8x - highlighting that best performance requires a memory overprovisioning of a factor of three or the workload suffers a substantial performance penalty. When the swap disk on each node is configured to a RamDisk, the overhead is reduced to a factor of 6.14x - which is still too high. Having observed the low overheads of RDMA in the MemcacheD example, it can be stipulated that sharing unused memory across the entire compute cluster instead of through a swap device to a remote memory location can further reduce the overhead. However the rapid allocation and deallocation of remote memory is imperative for the sharing of a memory pool to be effective.

c. Cassandra workload

The third experiment focuses on the impact of composable storage by using Cassandra, a popular persistent (i.e. disk based) key value NoSQL store as the workload. In the traditional setup (as shown in Fig. 12 (a)), a single server is populated with eight SATA disks that together form the block storage for a ZFS filesystem on which the key value pair storage resides. Ultimately the number of disks in the server is limited to the order of 10s due to constraints imposed by the packaging and the SATA v3 bandwidth, which is limited to 6 Gbps. In the composable setup (as shown in Fig. 12 (b)), there are a total of 4 storage nodes with eight disks attached to each node and Cassandra was accessed over a 10 Gbps Ethernet. The ZFS cache was limited and data was flushed out of the page cache to ensure that almost all accesses go to disk. A client consisting of 20 threads issued 10K operations (95% read) uniformly accessing the data domain.

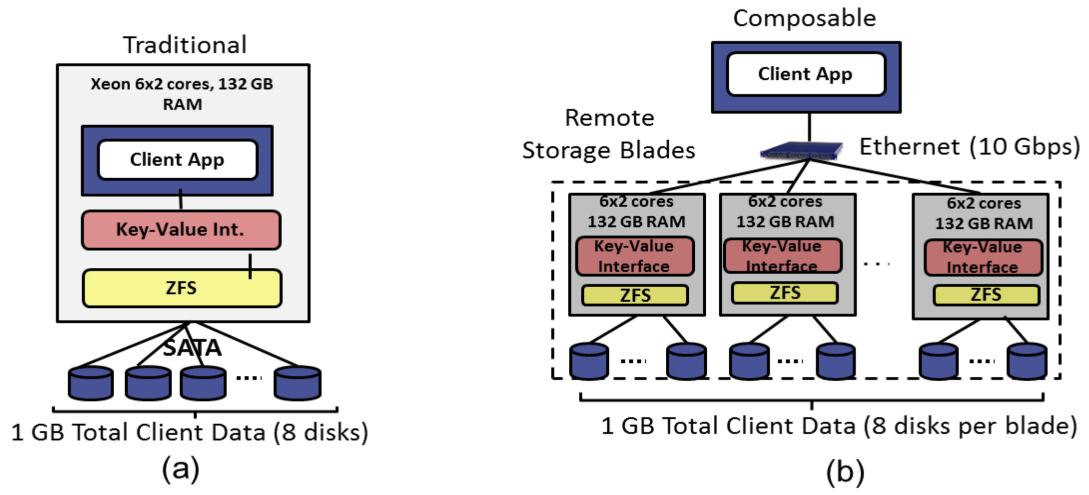

Figure 12: Experimental setup for (a) traditional vs. (b) composable HDDs for Cassandra workloads.

The bandwidth and latency improvement are shown in Figures 13 and 14. Access block size is set at 256KB and 512KB. For these block sizes, the throughput is improved up to 195% and 79 %, respectively, and latency improvement is 67% and 51%, respectively, for the composable system case. This experiment substantiates the thesis that accessing data from across multiple disks connected via Ethernet poses less of a bandwidth restriction than SATA and thus improves throughput and latency of data access and obviates the need for data locality. Overall, composable storage systems are cheaper to build, manage, and incrementally scalable, and offer superior performance than traditional setups.

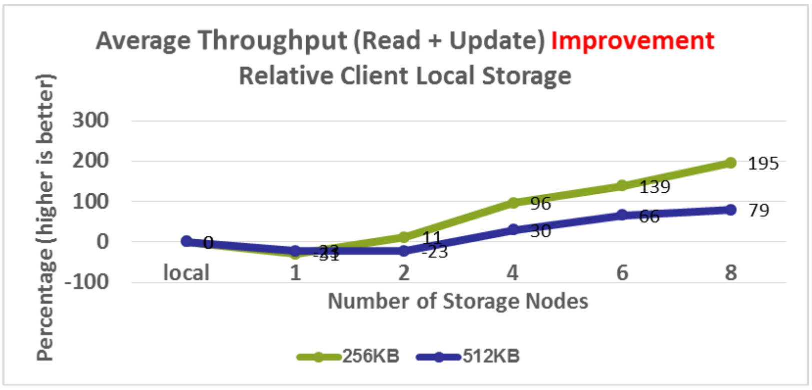

Figure 13: Throughput improvement of disaggregated storage for Cassandra workload

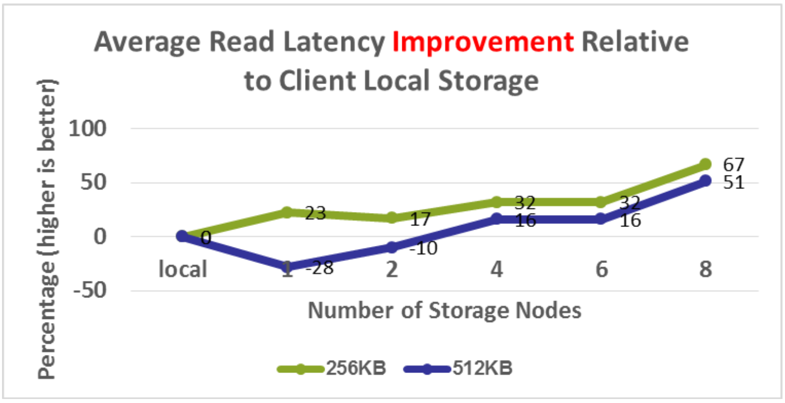

Figure 14: Latency improvement of disaggregated storage for Cassandra workload

10 SUMMARY AND FUTURE WORK

Deploying big data applications with large volume, high velocity, wide variety of modalities involving NoSQL, MapReduce, Spark/Hadoop in a cloud environment are facing the challenges of fast changing system configuration requirements due to highly dynamic workload constraints, varying innovation cycles of system hardware components, and the need for maximal sharing of systems and subsystems resources. Composable system offers the potential of addressing these challenges. Datacenters based on this architecture allows the refactoring of the datacenter for improved operating efficiency and decoupled innovation cycles among components while the datacenter network becomes the "backplane" of the datacenter.

In this paper, the feasibility of composable systems is demonstrated through building a number of rack scale composable system prototypes including one based on PCIe switch. Through empirical approaches, the opportunities and challenges for leveraging the composable architecture for rack scale cloud datacenters are evaluated with a focus on big data and NoSQL workloads. We also establish the implications and requirements for network and resource provisioning and management. Based on this assessment and experimental results, we conclude the following:

- A composable rack scale architecture with appropriate programming models and resource provisioning is likely to achieve improved datacenter operating efficiency. This architecture is particularly suitable for heterogeneous and fast evolving workload environments as these environments often have dynamic resource requirements and can benefit from the improved elasticity of the physical resource pooling offered by the composable rack scale architecture.
- Composable resources can be exposed through hardware based, hypervisor/operating system based, and middleware/application based approaches. Directly expose resource composability to applications and manage using application-level knowledge is likely to achieve the best flexibility and performance gain.
- The primary concern for the composable architecture is the potential performance impacts arising from accessing resources such as memory, GPU, and I/O from non-local shared resource pools. Retaining sufficient local DRAM serving as the cache for the pooled memory as opposed to full disaggregation of memory resources and retain no local memory for the CPU is always recommended to minimize the performance impact due to latency incurred from accessing remote memory. Higher SMT levels and/or explicit management by applications that maximize thread level parallelism are also essential to further minimize the performance impact.
- Negligible latency and throughput penalty are incurred in the MemcacheD experiments for the read/update operations if these operations are 75% local and the data size is 64 KB. Smaller data size results in larger latency penalty while larger data size results in larger throughput penalty when the ratio of nonlocal operations is increased to 50% and 75%.
- Frequent underutilization of memory is observed while CPU is more fully utilized across the cluster in the Giraph experiments. However, introducing composable system architecture in this environment is not straightforward as sharing memory resources among nodes within a cluster through configuring RamDisk presents very high overhead.

Consequently, it is stipulated that sharing unused memory across the entire compute cluster instead of through a swap device to a remote memory location is likely to be more promising in minimizing the overhead. In this case, rapid allocation and deallocation of remote memory is imperative to be effective.

- The Cassandra experiment substantiated the thesis that accessing data from across multiple disks connected via Ethernet poses less of a bandwidth restriction than SATA and thus improves throughput and latency of data access and obviates the need for data locality. Overall composable storage systems are cheaper to build, manage and incrementally scalable and offer superior performance than traditional setups.
- The experiments involving rack scale architecture using PCIe switch demonstrated the feasibility of PCIe based composable architecture where the I/O is composed dynamically from multiple nodes.

Our results support the importance of Big Data in the Cloud since the next-generation of services should be investigated to meet demands from volume, velocity and variety aspects of Big Data services. Our proposal can provide better technical performance and capacity for the future datacenters as demonstrated in the paper.

The next step of this work will include demonstrating in-memory Spark-based big data and NoSQL workloads in a composable system environment involving composable memory and GPU resources. We also plan to explore use workload cases from cybersecurity, cognitive computing, and internet of things.

REFERENCES

[1] L. A. Barroso, J. Clidaras and U. Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition, 2013.

[2] NIST SP 800-145, A NIST definition of cloud computing, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[3] NIST SP 500-292, Cloud Computing Reference Architecture, v1.0. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505

[4] C.-S. Li, B. L. Brech, S. Crowder, D. M. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, S. Pappe, B. Rajaraman, J. Rao, R. P. Ratnaparkhi, R. A. Smith and M. D. Williams. Software defined environments: An introduction. In IBM Journal of Research and Development Vol. 58 No. 2/3 pp. 1-11, March/May, 2014.

[5] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch (2012, October). Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In Proceedings of the Third ACM Symposium on Cloud Computing (p. 7). ACM.

[6] Q. Zhang, J. L. Hellerstein, and R. Boutaba. "Characterizing task usage shapes in Google's compute clusters." Large Scale Distributed Systems and Middleware Workshop (LADIS'11). 2011.

[7] A. Samih, R. Wang, C. Maciocco, T. Y. C. Tai, and Y. Solihin (2011, October). A collaborative memory system for high-performance and cost-effective clustered architectures. In Proceedings of the 1st Workshop on Architectures and Systems for Big Data (pp. 4-12). ACM.

[8] V. Chang. The business intelligence as a service in the cloud. Future Generation Computer Systems, 37, 512-534, 2014.

[9] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt and T. F. Wenisch. Disaggregated Memory for Expansion and Sharing in Blade Servers. In Proc. ISCA, 2009.

[10] PCI-SIG: Peripheral Component Interconnect Express (PCIe) 4.0 http://www.pcisig.com, 2015.

[11] IEEE P802.3bs 400 Gb/s Ethernet Task Force, http://www.ieee802.org/3/bs/

[12] IBM Corp., z/VM built on IBM Virtualization Technology General Information Version 4 Release 3.0, 2002.

[13] J. Jann, L. M. Browning, & R. S. Burugula (2003). Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers. *IBM Systems Journal*, *42*(1), 29-37, 2003.

[14] C. Xu, Y. Bai, and C. Luo. "Performance evaluation of parallel programming in virtual machine environment." Network and Parallel Computing, 2009. NPC'09. Sixth IFIP International Conference on. IEEE, 2009.

[15] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch. System-level implications of composable memory. In Proc. HPCA, 2012.

[16] GraphLab. http://graphlab.com/

[17] Memcached - a distributed memory object caching system. http://memcached.org/

[18] PigMix benchmark tool.http://cwiki.apache.org/confluence/display/PIG/PigMix

[19] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout (2011) May. It's Time for Low Latency. In HotOS (Vol. 13, pp. 11-11).

[20] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, (2013) Network support for resource disaggregation in next-generation datacenters. In Proc. HotNets.

[21] C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, J. Kepner, A. McCabe, P. Michaleas, J. Mullen, D. O'Gwynn, and A. Prout (2012) September. Driving big data with big compute. In High Performance Extreme Computing (HPEC), 2012 IEEE Conference on (pp. 1-6). IEEE.

[22] S. Ackermann, V. Jovanovic, T. Rompf, and M. Odersky (2012). Jet: An embedded DSL for high performance big data processing. In International Workshop on End-to-end Management of Big Data (BigData 2012) (No. EPFL-CONF-181673).

[23] R. Xin, U. C. AMPLab, J. Gonzalez, J. Rosen, M. Zaharia, M. Franklin, S. Shenker, and I. Stoica, (2013) Beating State-of-the-art By-10000%. In CIDR.

[24] M. Franklin (2013, October). The Berkeley Data Analytics Stack: Present and future. In Big Data, 2013 IEEE International Conference on (pp. 2-3). IEEE.

[25] M. Zaharia., M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica (2010, June). Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing (Vol. 10, p. 10).

[26] J. Taylor (2013) ARM and disaggregated Rack, Keynote, LCA 2013. https://www.youtube.com/watch?v=LRfsIMM1KjQ

[27] J. Parikh (2015) Facebook News, Keynote, Opencompute Summit.

[28] Cisco UCS M-Series Modular Servers. http://www.cisco.com/c/en/us/products/servers-unified-computing/ucs-m-series-modular-servers/index.html

[29] AMD Disaggregates the Server, Defines New Hyperscale Building Block. http://www.seamicro.com/sites/default/files/MoorInsights.pdf

[30] SeaMicro Technology Overview. http://seamicro.com/sites/default/files/SM_TO01_64_v2.5.pdf

[31] Intel, Facebook Collaborate on Future Datacenter Rack Technologies, http://newsroom.intel.com/community/intel_newsroom/blog/2013/01/16/intel-facebook-collaborate-on-future-data-center-rack-technologies, Jan. 2013.

[32] Open Compute Project. http://www.opencompute.org

[33] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Ortí, (2010, June). rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In High Performance Computing and Simulation (HPCS), 2010 International Conference on (pp. 224-231). IEEE.

[34] A. Vahdat, H. Liu, X. Zhao, and C. Johnson (2011, March). The emerging optical data center. In Optical Fiber Communication Conference (p. OTuH2). Optical Society of America.

[35] N. Farrington, G. Porter, P. C. Sun, A. Forencich, J. Ford, Y. Fainman, G. Papen, and A. Vahdat (2012) A demonstration of ultra-low-latency data center optical circuit switching. ACM SIGCOMM Computer Communication Review, 42(4), pp.95-96.

[36] A. Vahdat, "Delivering Scale Out Data Center Networking with Optics--Why and How." Optical Fiber Communication Conference. Optical Society of America, 2012.

[37] C. Reano, R. May, E. S. Quintana-Orti, F. Silla, J. Duato, A. J. Pena, Influence of InfiniBand FDR on the Performance of Remote GPU Virtualization, IEEE International Conference on Cluster Computing (CLUSTER), pp. 1-8, 2013.

[38] V. Chang and G. Wills. (2016). “A model to compare cloud and non-cloud storage of Big Data”. *Future Generation Computer Systems, 57, 56-76*.

[39] Arista, “Arista Networks Cloud Networking Portfolio”, https://www.arista.com/en/products/switches, accessed on 23 December, 2015.

[40] Mellanox Technologies, “InfiniBand Performance”, http://www.mellanox.com/page/performance_infiniband, accessed on 23 December, 2015.

[41] Calient “Calient S320 Datasheet”, http://www.calient.net/members-area/?redirect-to=/download/s320-optical-circuit-switch-datasheet/, accessed on 23 December, 2015.

[42] H3 Platforms http://www.h3platform.com:443/opencart/index.php?route=product/category&path=60

[43] J. Suzuki, Y. Hidaka, J. Higuchi, T. Baba, N. Kami, and T. Yoshikawa, (2010, August). Multi-root share of single-root I/O virtualization (SR-IOV) compliant PCI Express device. In 2010 18th IEEE Symposium on High Performance Interconnects (pp. 25-31). IEEE.

[44] Scalable Logic: Enhanced Networking in the AWS Cloud. http://blogs.scalablelogic.com/2013/12/enhanced-networking-in-aws-cloud.html, 2013.

[45] Scalable Logic: Enhanced Networking in the AWS Cloud - Part 2. http://blogs.scalablelogic.com/2014/01/enhanced-networking-in-aws-cloud-part-2.html, 2014.

[46] S. Balakrishnan, R. Black, A. Donnelly, P. England, A. Glass, D. Harper, S. Legtchenko, A. Ogus, E. Peterson, and A. Rowstron, (2014) Pelican: A building block for exascale cold data storage. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14) (pp. 351-365).

[47] K. Spafford, J. S. Meredith, and J. S. Vetter. "Quantifying numa and contention effects in multi-gpu systems." Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units. ACM, 2011.

[48] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, and S. Yalamanchili (2011) Keeneland: Bringing heterogeneous GPU computing to the computational science community. Computing in Science and Engineering, 13(5), pp.90-95.

[49] P. Micikevicius "Multi-GPU programming." GPU Computing Webinars, NVIDIA (2011).

[50] T. P. Morgan (2015) “Facebook to open up Custom Machine Learning Iron” http://www.nextplatform.com/2015/12/10/facebook-to-open-up-custom-machine-learning-iron/

[51] T. A. Gregg, D. Craddock, D. J. Stigliani, F. E. Bosco, E. E. Cruz, M. F. Scanlon, P. Sciuto, G. Bayer, M. Jung, and C. Raisch (2012) Overview of IBM zEnterprise 196 I/O subsystem with focus on new PCI express infrastructure. IBM Journal of Research and Development, 56(1.2), pp.8-1.

[52] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In Proceedings of the 7th international conference on Autonomic computing, pp. 11–20. ACM, 2010.

[53] 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, page 7. ACM, 2012.

[54] A. Samih, R. Wang, C. Maciocco, T.-Y. C. Tai, and Y. Solihin. A collaborative memory system for high-performance and cost-effective clustered architectures. In Proceedings of the 1st Workshop on Architectures and Systems for Big Data, pages 4–12. ACM, 2011.

[55] Q. Zhang, J. L. Hellerstein, and R. Boutaba. Characterizing task usage shapes in Google compute clusters. In Large Scale Distributed Systems and Middleware Workshop (LADIS.11), 2011.

[56] B. Abali, R. J. Eickemeyer, H. Franke, C.-S. Li, and M. A. Taubenblatt (2015). Disaggregated and optically interconnected memory: when will it be cost effective? *arXiv preprint arXiv:1503.01416*, arXiv:1503.01416 , 3 Mar 2015.

[57] D. M. Tullsen and J. A. Brown (2001, December). Handling long-latency loads in a simultaneous multithreading processor. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture* (pp. 318-327). IEEE Computer Society.

[58] SPEC. SPEC CPU 2006 benchmarks. http://spec.org

[59] C. Chou, A. Jaleel, and M. K. Qureshi. "CAMEO: A Two-Level Memory Organization with Capacity of Main Memory and Flexibility of Hardware-Managed Cache." *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014

[60] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D Joseph, R. H Katz, S. Shenker, I. StoicaMesos (2011): Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. Proc. ACM USENIX Symposium on Networked Systems Design & Implementation (NSDI), 2011.

[61] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes (2013, April). Omega: flexible, scalable schedulers for large compute clusters. In Proceedings of the 8th ACM European Conference on Computer Systems (pp. 351-364). ACM.

[62] N. Roy, A. Dubey, and A. Gokhale. "Efficient autoscaling in the cloud using predictive models for workload forecasting." Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011.

[63] A. Khoshkbarforoushha, R. Ranjan, R. Gaire, P. P. Jayaraman, J. Hosking, and E. Abbasnejad (2015). Resource Usage Estimation of Data Stream Processing Workloads in Datacenter Clouds. *arXiv preprint arXiv:1501.07020*.

[64] B. Mozafari, C. Curino, and S. Madden (2013, January). DBSeer: Resource and Performance Prediction for Building a Next Generation Database Cloud. In *CIDR*.

[65] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy (2008, December). Profiling and modeling resource usage of virtualized applications. In*Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (pp. 366-387). Springer-Verlag New York, Inc..

[66] A. Ganapathi, Y. Chen, A. Fox, and R. Katz, and D. Patterson. (2010, March). Statistics-driven workload modeling for the cloud. In Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on (pp. 87-92). IEEE.

[67] C. Mitchell, Y. Geng, and J. Li. "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store." USENIX Annual Technical Conference. 2013.

[68] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis (2013) Mizan: a system for dynamic load balancing in large-scale graph processing, In Proc. Of Eurosys

[69] J. Jiang, J. Lu, G. Zhang, and G. Long (2013, May). Optimal cloud resource auto-scaling for web applications. In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on (pp. 58-65). IEEE.

[70] R. Cushing, S. Koulouzis, A. S. Belloum, and M. Bubak (2011, December). Prediction-based auto-scaling of scientific workflows. In Proceedings of the 9th International Workshop on Middleware for Grids, Clouds and e-Science (p. 1). ACM.

[71] M. Mao and M. Humphrey (2011, November). Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (p. 49). ACM.