dReDBox: a Disaggregated Architectural Perspective for Data Centers

Nikolaos Alachiotis[1], Andreas Andronikakis[1], Orion Papadakis[1], Dimitris Theodoropoulos[1], Dionisios Pnevmatikatos[1], Dimitris Syrivelis[2], Andrea Reale[2], Kostas Katrinis[2], George Zervas[3], Vaibhawa Mishra[3], Hui Yuan[3], Ilias Syrigos[4], Ioannis Igoumenos[4], Thanasis Korakis[4], Marti Torrents[5], and Ferad Zyulkyarov[5]

[1] Foundation for Research and Technology - Hellas
[2] IBM Research - Ireland
[3] University College London
[4] University of Thessaly
[5] Barcelona Supercomputing Center

Abstract. Data centers are currently defined by the physical infrastructure, leading to suboptimal utilization of resources and increased energy requirements. The dReDBox (disaggregated Recursive Datacenter in a Box) project addresses the problem of fixed resource proportionality in next-generation, low-power data centers by proposing a paradigm shift toward finer resource allocation granularity, where the unit is the function block rather than the mainboard tray. This introduces various challenges at the system design level, requiring elastic hardware architectures, efficient software support and management, and programmable interconnect. Hardware accelerators can be dynamically assigned to processing units to boost application performance, while high-speed, low-latency electrical and optical interconnect is a prerequisite for realizing the concept of data center disaggregation. This chapter presents the dReDBox hardware architecture and discusses design aspects of the software infrastructure for resource allocation and management. Furthermore, initial simulation and evaluation results for accessing remote, disaggregated memory are presented, employing benchmarks from the Splash-3 and the CloudSuite benchmark suites.

1 Introduction

Data centers have been traditionally defined by the physical infrastructure, imposing a fixed ratio of resources throughout the system. A widely adopted design paradigm assumes the mainboard tray, along with its hardware components, as the basic building block, requiring the system software, the middleware, and the application stack to treat it as a monolithic component of the physical system. The proportionality of resources per mainboard tray, however, which is set at design time, remains fixed throughout the life time of a data center, imposing various limitations to the overall data center architecture. First, the proportionality

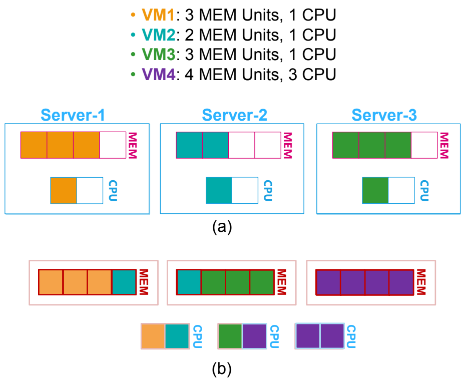
• VM1: 3 MEM Units, 1 CPU
• VM2: 2 MEM Units, 1 CPU
• VM3: 3 MEM Units, 1 CPU
• VM4: 4 MEM Units, 3 CPU
Server-1
Server-2
Server-3
MEM
CPU
(a)
(b)

Fig. 1. A limitation of current data center infrastructures regarding resource utilization (a), and the respective resource allocation scheme of dReDBox (b).

of resources at system level inevitably follows the fixed resource proportionality of the mainboard tray. With the mainboard tray as the basic, monolithic building block, system-level upgrades to facilitate increased resource requirements for additional memory, for instance, bring along parasitic capital and operational overheads that are caused by the rest of the hardware components on a tray, e.g., processors and peripherals. Second, the process of assigning/allocating resources to Virtual Machines (VMs) is significantly restricted by the physical boundaries of the mainboard tray, with respect to resource quantities. Such lack of flexibility in allocating resources to VMs eventually brings the overall system to a state of stranded resource capacity, while yielding the data center insufficient to facilitate the requirements of additional VMs. Third, technology upgrades need to be carried out on a per-server basis, raising significantly the costs when applied at scale, e.g., in an internet-scale data center that can potentially comprise thousands of servers.

However, as current data center systems are composed by a networked collection of monolithic building blocks, shifting toward an architectural paradigm that overcomes the aforementioned fixed proportionality of resources by breaking the boundaries of the motherboard tray to achieve finer granularity in resource allocation to VMs entails various challenges and open problems to address. The challenges and requirements broadly relate to four categories, namely the hardware platform, the memory, the network, and the system software. A representative hardware-platform-level requirement, for instance, entails the need to establish intra- and inter-tray interconnection paths that are programmable, yet introduce minimal communication overhead. For this purpose, a low-latency network architecture that is scalable and achieves high bandwidth is needed. Remote memory allocation support and management is required, as well as efficient ways to maintain coherency and consistency, while minimizing remote-memory access latency. Dedicated mechanisms to support dynamic on-demand network connectivity and scalability, as well as orchestration software tools that define resource topologies, generate and manage VMs, and ensure reliability and correctness while exploring optimizations, are prerequisites for the success of the overall approach. Fine-grained power management at the component-level is required in order to minimize overall data center energy consumption.

To this end, the dReDBox (disaggregated Recursive Datacenter in a Box) project aims at overcoming the issue of fixed resource proportionality in next generation, low-power data centers by departing from the mainboard-as-a-unit paradigm and enabling disaggregation through the concept of function-block-as-a-unit. The following section (Section 2) provides an overview of existing data center architectures and related projects, and presents the dReDBox approach to data center architecture design. The remaining sections are organized as follows. Sections 3 and 4 present the system architecture and the software infrastructure, respectively. Section 5 describes a custom simulation environment for disaggregated memory and present a performance evaluation. Finally, Section 6 concludes this chapter.

2 Disaggregation and the dReDBox Perspective

The concept of data center disaggregation regards resources as independent homogeneous pools of functionalities across multiple nodes. The increasingly recognized benefits of this design paradigm have motivated various vendors to adopt the concept, and significant research efforts are currently conducted toward that direction, both industrial and academic ones.

From a storage perspective, disaggregation of data raises a question regarding where should data reside, at the geographical level, to achieve short retrieval times observed by the end users, data protection, disaster recovery and resiliency, as well as to ensure that mission-critical criteria are met at all times. Various industrial vendors provide disaggregated solutions for that purpose, enabling flash capacity disaggregation across nodes, for instance, or the flexible deployment and management of independent resource pools. Klimovic *et al.* [13] examine disaggregation of PCIe-based flash memory as an attempt to overcome overprovisioning of resources that is caused by the existing inflexibility in deploying data center nodes. The authors report a 20% drop in application-perceived throughput due to facilitating remote flash memory accesses over commodity networks, achieving, however, highly scalable and cost-effective allocation of processing and flash memory resources.

A multitude of research efforts have focused on disaggregated memory with the aim to enable scaling of memory and processing resources at independent growth paces. Lim *et al.* [14, 15] present the “memory blade” as an architectural approach to introduce flexibility in memory capacity expansion for an ensemble of blade servers. The authors explore memory-swapped and block-access solutions for remote access, and address software- and system-level implications by developing a software-based disaggregated memory prototype based on the Xen hypervisor. They find that mechanisms which minimize the hypervisor overhead are preferred in order to achieve low-latency remote memory access.

Cheng-Chun Tu *et al.* [24] present Marlin, a PCIe-based rack area network that supports communication and resource sharing among disaggregated racks. Communication among nodes is achieved via a fundamental communication primitive that facilitates direct memory accesses to remote memory at the hardware level, with PCIe and Ethernet links used for inter-rack and intra-rack communication, respectively. Dragojevic *et al.* [9] describe FaRM, a main memory distributed computing platform that, similarly to Marlin, relies on hardware-support for direct remote memory access in order to reduce latency and increase throughput.

Acceleration resources, e.g., FPGAs, are increasingly being explored to boost application performance in data center environments. Chen *et al.*[8] present a framework that allows FPGA integration into the cloud, along with a prototype system based on OpenStack, Linux-KVM, and Xilinx FPGAs. The proposed framework addresses matters of resource abstraction, resource sharing among threads, interfacing with the underlying hardware, and security of the host environment. Similarly, Fahmy [10] describe a framework for accelerator integration in servers, supporting virtualized resource management and communication. Hardware reconfiguration and data transfers rely on PCIe, while software support that exposes a low-level API facilitates FPGA programming and management. Vipin *et al.* [25] present DyRACT, an FPGA-based compute platform with support for partial reconfiguration at runtime using a static PCIe interface. The DyRACT implementation is targeting Virtex 6 and Virtex 7 FPGAs, while a video-processing application that employs multiple partial bitstreams is used as a case study for validation and evaluation purposes.

More recently, Microsoft presented the Configurable Cloud architecture [7], which introduces reconfigurable devices between network switches and servers. This facilitates the deployment of remote FPGA devices for acceleration purposes, via the concept of a global pool of acceleration resources that can be employed by remote servers as needed. This approach to disaggregation eliminates the, otherwise, fixed one-to-one ratio between FPGAs and servers, while the particular FPGA location, between the server and the network switches, enables the deployment of reconfigurable hardware for infrastructure enhancement purposes, e.g., for encryption and decryption. A prior work by Microsoft, the Catapult architecture [21], relied on a dedicated network for inter-FPGA communication, therefore raising cabling costs and management requirements. Furthermore, efficient communication among FPGAs was restricted to a single rack, with software intervention required to establish inter-rack data transfers.

EU-funded research efforts, such as the Vineyard [12] and the ECOSCALE [17] projects, aim at improving performance and energy efficiency of compute platforms by deploying accelerators. The former addresses the problem targeting data center environments, via the deployment of heterogeneous systems that rely on data-flow engines and FPGA-based servers, while the latter adopts a holistic approach toward a heterogeneous hierarchical architecture that deploys

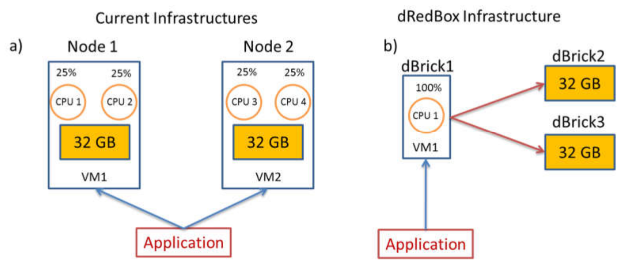

Fig. 2. Example of a memory-intensive application and the respective resource allocation schemes in a current infrastructure (a) and dReDBox (b).

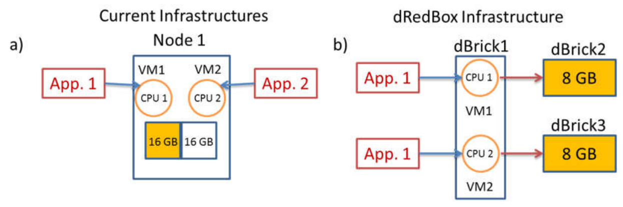

Fig. 3. Example of a compute-intensive application and the respective resource allocation schemes in a current infrastructure (a) and dReDBox (b).

accelerators, along with a hybrid programming environment based on MPI and OpenCL.

The dReDBox approach aims at providing a generic data center architecture for disaggregation of resources of arbitrary types, such as processors, memories, and FPGA-based accelerators. Basic blocks, dubbed bricks, construct homogeneous pools of resources, e.g., a compute pool comprising multiple compute bricks, with system software and orchestration tools implementing software-defined virtual machines which exhibit customized amounts of resources that better serve application needs. High-speed, low-latency optical and electrical networks will establish inter- and intra-tray communication among bricks, respectively. Figures 2 and 3 illustrate the expected resource allocation schemes, enabled by the dReDBox infrastructure, for serving the requirements of memory- and compute-intensive applications, respectively. The following section presents the proposed dReDBox hardware architecture.

3 System Architecture

This section presents the overall architecture of a dReDBox data center. Multiple dReDBox racks, interconnected via an appropriate data center network, form a dReDBox data center. This overall architecture is shown in Figure 4. The dReDBox architecture comprises pluggable compute/memory/accelerator modules (termed "bricks" in dReDBox terminology) as the minimum field-replaceable units. A single or sets of multiples of each brick type forms an IT resource pool of the respective type. A mainboard tray with compatible brick slots and on-board electrical crossbar switch, flash storage and baseboard management components is used to support up to 16 bricks. A 2U carrier box (dBOX, visually corresponding from the outside to a conventional, rack-mountable data center server) in turn hosts the mainboard tray and the intra-tray optical switch modules.

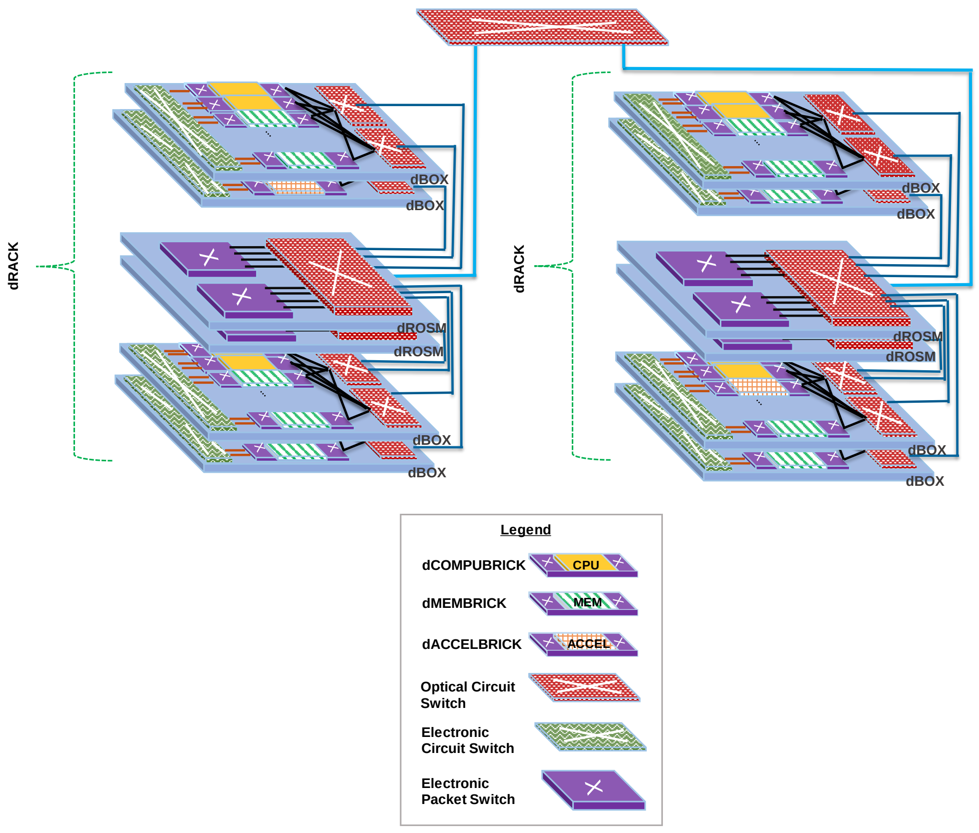

Fig. 4. Overview of a dReDBox rack architecture comprising several dBOXes interconnected with hybrid optical and electrical switching (dROSM).

3.1 The dBRICK Architecture

The dBRICK is the smallest realization unit in the dReDBox architecture. The term encompasses general-purpose processing (dCOMPUBRICK), randomaccess memory (dMEMBRICK), and application-specific accelerators (dACCELBRICK). As described above, dBRICKs will be connected to the rest of the system by means of a tray that, besides connectivity, will also provide the necessary power to each brick.

Compute Brick Architecture (dCOMPUBRICK) The dReDBox compute brick (Fig. 5) is the main processing block in the system. It hosts local

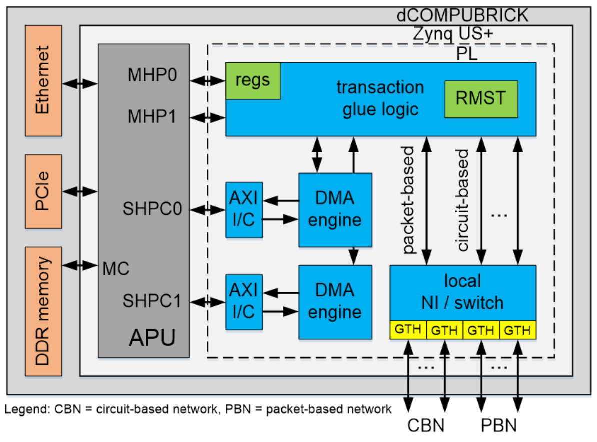

Fig. 5. Block diagram of a dCOMPUBRICK. The MPSoC integrates an Application Processing Unit (APU) for software execution. The on-chip programmable logic on the SoC is used to host transaction glue logic, housekeeping state, and communication logic, required for accessing disaggregated resources. The local DMA engines allow the system software to efficiently migrate pages from remote memory regions to local DDR memory.

off-chip memory (DDR4) for low-latency and high-bandwidth instruction read and read/write data access, as well as Ethernet and PCIe ports for data and system communication and configuration. Also, each dCOMPUBRICK features QSPI-compatible flash storage (16-32 MB) and a micro-SD card socket (not shown in Fig. 5) to facilitate reset and reconfiguration of the brick in the case of disconnection, as well as for debugging purposes.

The compute brick can reach disaggregated resources, such as memory and accelerators, via dReDBox-specific glue intellectual property (termed "Transaction Glue Logic") on the datapath and communication endpoints implemented on the programmable logic of the dCOMPUBRICK MPSoC. System interconnection to disaggregated resources occurs via multiple ports leading to circuit-switched tray- and rack-level interconnects. As also shown in Figure 5, we also experiment with packet-level system/data interconnection, using Network Interface (NI) and a brick-level packet switch (also implemented on programmable logic of the dCOMPUBRICK MPSoC), on top of the inherently circuit-based interconnect substrate. There is potential value in such an approach, specifically in terms of increasing the connectivity of a dCOMPUBRICK due to multi-hopping and thus creating an opportunity to increase the span of resource pools reachable from a single dCOMPUBRICK.

Memory Brick Architecture (dMEMBRICK) Figure 6 illustrates the memory brick (dMEMBRICK) architecture, which is a key disaggregation feature of dReDBox. It will be used to provide a large and flexible pool of memory resources which can be partitioned and (re)distributed among all processing nodes (and corresponding VMs) in the system. dMEMBRICKs can support multiple links. These links can be used to provide higher aggregate bandwidth, or can be partitioned by the orchestrator and assigned to different dCOMPUBRICKs, depending on the resource allocation policy used. This functionality can be used in two ways. First, the nodes can share the memory space of the dMEMBRICK, implementing essentially a shared memory block (albeit shared among a limited number of nodes). Second, the orchestrator can also partition the memory of the dMEMBRICK, creating private "partitions" for each client. This functionality allows for fine-grained memory allocation. It also requires translation and protection support in the glue logic (transaction glue logic block) of the dMEMBRICK.

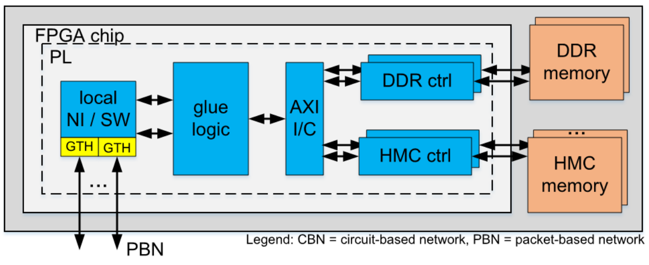

Fig. 6. dMEMBRICK architecture featuring the Xilinx Zynq Ultrascale+ MPSoC (EG version); the local switch forwards system/application data to the memory brick glue logic, which interfaces different memory module technologies.

The glue logic implements memory translation interfaces with the requesting dCOMPUBRICKs, both of which are coordinated by the orchestrator software. Besides network encapsulation, the memory translator, managed by orchestrator tools, controls the possible sharing of the memory space among multiple dCOMPUBRICKs, enabling support for both sharing among and protection between dCOMPUBRICKs. The control registers allow the local mapping of external requests to local addresses to allow more flexible mapping and allocation of memory.

Acceleration Brick Architecture (dACCELBRICK) A dACCELBRICK hosts accelerator modules that can be used to boost application performance based on a near-data processing scheme [20]; instead of transmitting data to remote dCOMPUBRICKs, certain calculations can be performed by local accelerators, thus improving performance while reducing network utilization.

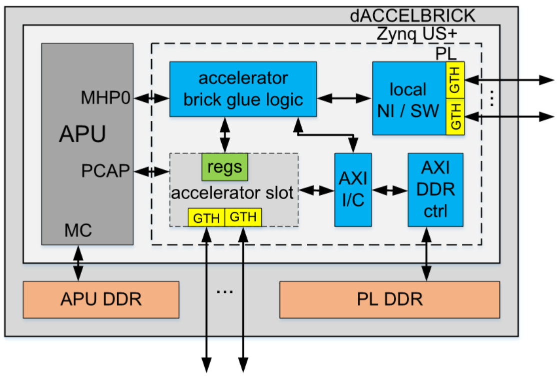

Fig. 7. The dACCELBRICK architecture for accommodating application-specific accelerators.

Figure 7 depicts the dACCELBRICK architecture. The dACCELBRICK consists of the dynamic and the static infrastructure. The dynamic infrastructure consists of a predefined, reconfigurable slot in the PL that hosts hardware accelerators. As depicted in Figure 7, the accelerator wrapper template integrates a set of registers that can be accessed by the glue logic to monitor and control (e.g., debug) the accelerator. Moreover, the wrapper provides a set of high-speed transceivers (e.g., GTHs) for direct communication between the accelerator and other external resources. Finally, an AXI-compatible port interfaces directly with an AXI DDR controller, allowing the hardware accelerator to utilize the local PL DDR memory during data processing. The static infrastructure hosts all required modules for (a) supporting dynamic hardware reconfiguration, (b) interfacing with the hardware accelerator, and (c) establishing communication with remote dCOMPUBRICKs. To support hardware reconfiguration, in the current implementation, the local APU executes a "thin" middleware responsible for (a) receiving bitstreams from remote dCOMPUBRICKs (through the accelerator brick glue logic), (b) storing bitstreams in the APU DDR memory, and (c) reconfiguring the PL with the required hardware IP via the PCAP port. To monitor/control the hardware accelerator, the glue logic can read/write the wrapper registers. In addition, the glue logic interfaces with the local NI/switch for data transfers between the dACCELBRICK and remote dCOMPUBRICKs.

3.2 The dTRAY Architecture

dTRAYs may be composed of arbitrary combinations of the three different types of dBRICKs detailed above. A dTRAY will have standard 2U size and may contain up to 16 bricks. It is expected that the number of dMEMBRICKs will be larger than the number of dCOMPUBRICKs and dACCELBRICKs, since a dCOMPUBRICK is expected to access multiple dMEMBRICKs. The different dBRICKs are interconnected among each other within the dTRAY and also with other dBRICKs from different dTRAYs. Figure 8 illustrates the dTRAY architecture. Four different networks, a low-latency high-speed electrical network, an Ethernet network, a low-latency high-speed optical network, and a PCIe network, will provide connectivity between the different bricks. Accessing remote memory will use both optical and electrical low-latency, high-speed networks. Accesses to remote memory placed in a dMEMBRICK within a dTRAY will be implemented via an electrical circuit crossbar switch (dBESM in Figure 4 is labeled as High Speed Electrical Switch) and will connect directly to the GTH interface ports available on the programmable logic of the bricks. The dBESM switch will have 160 ports. This is the largest dBESM switch available on the market today supporting our speed requirements. The latency will be as low as 0.5ns and the bandwidth per port will be 12Gbps. This network will be used for intra-tray memory traffic between different bricks inside the tray. dBESM will not be used for inter-tray memory traffic due the limitations of the electrical communication in larger distances (latency). In addition, using electrical network for intra-tray communication instead of an optical network would not require signal conversion from electrical to optical and vice versa and thus it will be lower latency and lower power consumption. The optical network on the dTRAY, providing inter-tray connectivity, will be implemented with multiple optical switch modules (dBOSM in dReDBox terminology). Each dBOSM switch will have 24 optical ports. The latency of the dBOSM optical switch would be around 5ns and the bandwidth would be in the range of 384Gbps. dBRICKs will connect to the dBOSM via GTH interface ports available on the programmable logic of the SoC. The GTH bandwidth is 16Gbps. A total of 24 GTH ports will be available in the SoC, with 8 of them used to connect the SoC to the dBOSM. On a fully populated tray hosting 16 bricks, a maximum of 256 optical ports may be used to fully interconnect the bricks of each tray. A Mid-Board Optics (MBO) device mounted on each dBRICK will be used to convert the electrical signals coming from the GTH ports and aggregate them into a single fibre ribbon; the other end of the ribbon will be attached to a local dBOX's dBOSM optical switch. Each MBO supports up to eight ports. The dBRICKs will use 10 GTH ports to connect to dBOSM. The number of GTH's per SoC connecting to the dBOSM is limited by the size of the dBOSM. A 160-port dBOSM can support a maximum of 10 GTH per dBRICK, given a maximum of 16 dBRICKs on a tray. An Ethernet (ETH) network will be used for regular network communication and board management communication (BMC). The bandwidth will be 1Gbps and it will have a hierarchical topology. Bricks on the same tray will interconnect via a PCIe interface. Inter-brick interconnection between bricks on different trays

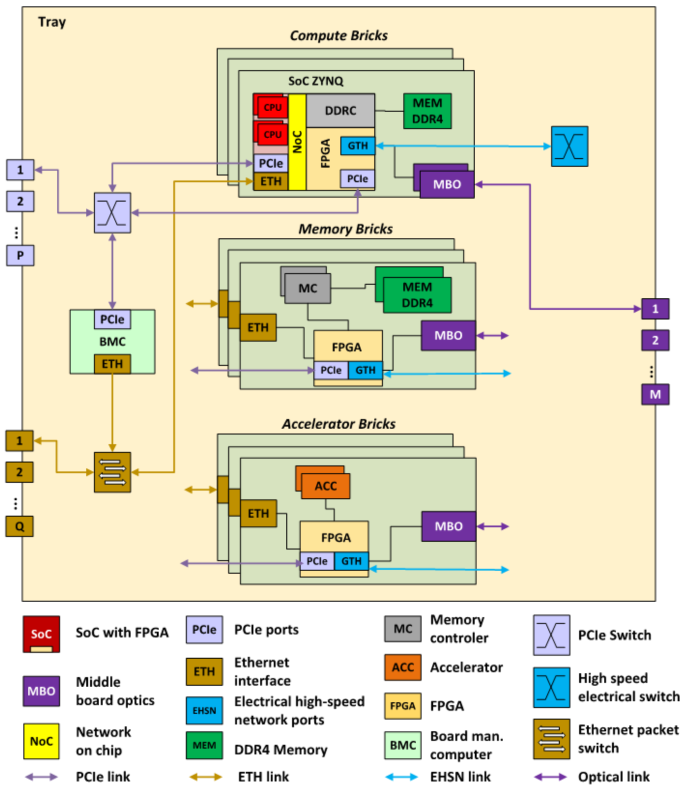

Fig. 8. Sample tray architecture with multiple bricks interconnected through optical and electrical interconnection networks.

within the same rack will be provided via a PCIe switch, which will exit the tray with one or more PCIe cable connectors. The PCIe interface will be used for signalling and interrupts, as well as for attachment to remote peripherals. This network can also be used to (re)configure the FPGAs in each SoC.

3.3 The dRACK Architecture

Figure 4 introduced the high-level dRACK architecture of the dReDBox architecture. Multiple dTRAYs of different configurations can be placed in the same dRACK. These dTRAYs can feature different proportions of Compute, Memory, and Accelerators. dRACKs are organized into dCLUSTs due to a restriction that the largest dROSM (rack switch) will not be able to interconnect all the optical links from the dTRAYs, as well as for facilitating management.

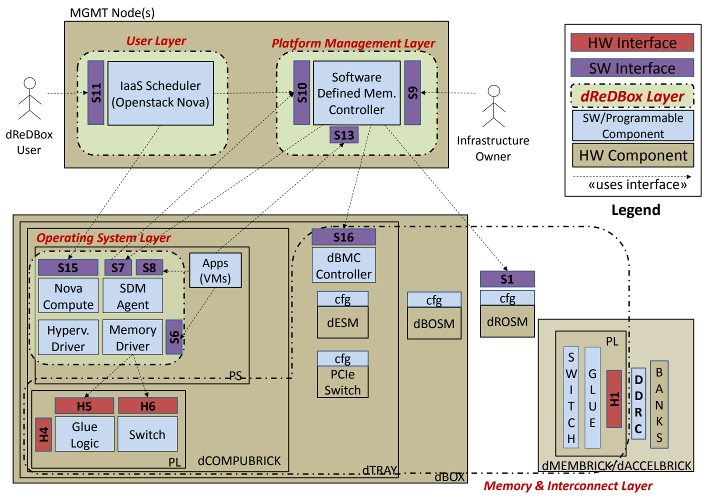

Fig. 9. The three dReDBox layers for software support. The user layer provides the end-user interface. The resource management layer enables management and monitoring of resources. The operating system layer on the compute bricks provides hypervisor support to manage virtual machines.

4 Software Infrastructure

In this section, the overall software architecture and interactions between the software components, enabling the disaggregated nature of the dReDBox platform, are presented. Software deployment expands to various parts of the system and thus a hierarchical design is necessary in order to reduce complexity and facilitate implementation. Figure 9 shows the division of software components into three layers that range from the interaction with the user and the handling of virtual machine deployment requests, to platform management, to interconnection paths synthesis and, eventually, to the necessary extensions of the operating system that enable support for remote memory access.

4.1 User Layer

This is the layer that serves as the interface between the end user and the system, allowing to reserve virtualized resources and to deploy virtual machines. In addition, it allows the system administrator to register hardware components and monitor them via a graphical user interface.

IaaS Scheduler dReDBox takes advantage of OpenStack [2], the de facto standard cloud operating system, for providing an Infrastructure-as-a-Service platform to provision and manage virtual resources. However, various extensions are required in order to comply with the disaggregated architecture of the dReDBox platform.

OpenStack, through its Compute service, Nova [4], employs the scheduler to select the most qualified host in terms of user requirements for compute and memory capacity. This is achieved by filtering and assigning weights to the list of available hosts. In traditional cloud computing systems, resources are provided only by the compute nodes (hosts) and, as a result, scheduling decisions and resource monitoring are limited by this restriction. To take full advantage of disaggregated resources, e.g., memory and accelerators, we modify Nova Scheduler by implementing a communication point with the resource management layer, described below, through a REST API. We, thus, intersect the workflow of virtual machine scheduling and retrieve a suitable compute node for the deployment of a user's virtual machine in order to maximize node's utilization. Furthermore, in the case that the amount of allocated memory to the host does not satisfy user requirements, the resource management layer proceeds by allocating additional remote memory and establishing the required connections between the bricks. Besides virtual machine scheduling, we extend Openstack's web user interface, Horizon [3], in order to enable the administrator to monitor compute bricks, deployed virtual machines, and allocated remote resources. Finally, administrators, through this extended interface, can also register new components (compute bricks, memory modules etc) and the physical connections between them.

4.2 Resource Management Layer

This is the layer where the management and monitoring of the availability of the various system resources occurs. More specifically, the current state of memory and accelerator allocations is preserved along with the dynamic configurations of the hardware components interconnecting bricks (optical switches, electrical crossbars, etc).

Software Defined Memory Controller The core entity of the resource management software is the Software Defined Memory (SDM) Controller. It runs as a separate and autonomous service, implemented in Python programming language and exposes REST APIs for the interaction with both the IaaS Scheduler and the agents running on compute bricks. Its primary responsibility is to handle allocation requests for both memory and accelerators. Memory requests are arriving to the REST interface from the IaaS scheduler, which is requesting the reservation of resources upon the creation of a virtual machine with a pre-defined set of requirements for vCPU cores and memory size, called "flavor". The SDM Controller, then, returns a list of the candidate compute bricks that can satisfy both compute and memory requirements after the consideration of total memory availability in the system and connection points on compute bricks (transceiver ports) and switches (switch ports). Allocation algorithms aiming at improving power savings and/or performance, through low latency memory accesses, are

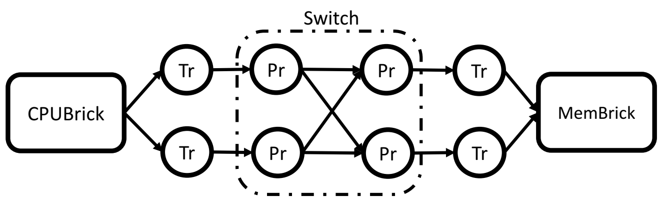

Fig. 10. Graph-based resource modeling example for a system that comprises a ComputeBrick, a MemoryBrick, two transceivers per brick, and one switching layer. Bricks, Transceivers and Switch ports are depicted as vertices.

employed to select the most suitable memory brick for remote memory allocation.

After the selection of the bricks, where the deployment and the memory allocation will take place, a subsequent component, part of the SDM Controller, called platform synthesizer, is employed. Its main function is to maintain an overview of the whole platform hardware deployment and to drive the orchestration and the dynamic generation of system interconnect configurations. From there, these configurations are passed through REST calls to the interfaces managing the electrical crossbars and the optical circuit switches as well as to the agents running on compute bricks, so that they can configure the programmable logic and initiate hotplugging of remote memory segments.

Graph Database The representation of resources and interconnection between them is realized as a directed graph structure implemented in JanusGraph [1] distributed graph database. Resources are modeled as vertices, while the edges between them represent physical connections. Figure 10 illustrates a simple example. More specifically, compute/memory/accelerator bricks, as well as transceivers and switch ports are modeled as vertices. We then use directed edges to connect all the transceivers that are located on a compute brick (acting as root) to the transceivers located on memory bricks (acting as sinks). This happens despite the fact that transceivers are bidirectional, but for the sake of convenience we consider transceivers belonging to compute bricks as logical output and transceivers belonging to memory and accelerator bricks as logical input. One or more switching layers exist between transceivers on different bricks. At the first layer, brick (compute/memory/accelerator) transceiver ports are physically connected either to the ports of the electrical crossbar or to the ports of the optical circuit switch (dBOSM), both of which are residing on the same dBOX as the brick. Remaining ports of the dBOSM are physically connected to the next switching layer which is the rack switch (dROSM). Switch ports of the same switch are fully interconnected in order to make the establishment of a circuit between any of the switch ports possible. Traversing the edges of the graph that originate at a compute brick and finish at memory bricks allows to retrieve the required paths, i.e., the switch ports that need to be configured for the establishment of the connection between the bricks. After the successful configuration of the components participating in the interconnection, we increase a usage counter defined as a vertex property on each one, denoting that it is already part of the allocation and that it should not be reused for interconnecting additional bricks. Several techniques to avoid loops in traversals and to improve performance have been used, which, however, are beyond the scope of this chapter.

4.3 Compute Brick Operating System Layer

The compute brick runs the hypervisor, a modified version of the Linux kernel with KVM modules that add virtual machine management capabilities. In this section, we describe the main operating-system-level extensions over a traditional Linux system. As expected, innovations in comparison with state-of-the-art systems relate to the management of remote memory and the allocation/deallocation process.

Memory Driver The memory driver is a collection of hypervisor-level modules supporting dynamic allocation and deallocation of memory resources to the host operating system running on a compute brick. The memory driver implements interfaces which configure the remote memory access, both during guest VM allocation and for dynamic resizing of remote resources. In the rest of this section, we focus on the specification of the main sub-components of the memory driver, namely, memory hotplug and NUMA extensions.

Once the required hardware components are set up to connect a compute brick with one or more remote memory bricks, the hypervisor running on the compute brick makes the new memory available to its local processes. These processes include VMs, each living in a distinct QEMU process. In order to make remote memory available, the hypervisor extends its own physical address space. This extended part corresponds to the physical addresses that are mapped to remote destinations. Once these addresses are accessed by the local processor, they are intercepted by the programmable logic and forwarded to the appropriate destination.

Memory hotplug is a mechanism that was originally developed to introduce software-side support for server boards that allow to physically plug additional memory SO-DIMMs at runtime. At insertion of new physical memory, a kernel needs to be notified about its existence and subsequently to initialize corresponding new page frames on the additional memory as well as to make them available to new processes. In memory hotplug terminology, this procedure is referred to as a “hot add”. Similarly, a “hot remove” procedure is triggered via software to allow detaching pages from a running kernel and to allow the physical removal of memory modules. While originally developed for the x86 architecture, memory hotplug has been ported so far to several different architectures, with our implementation focusing on ARM64. The compute-brick kernel reuses the functionalities provided by memory hotplug to extend the operating system's physical memory space by adding new pages to a running kernel and, similarly, removing them once memory is deallocated from the brick. Unlike the standard use of memory hotplug in traditional systems, there is no physical attachment of new hardware in dReDBox; for this reason, both hot add and hot remove have to be initiated via software in response to remote memory attach/detach requests.

Although the choice of building remote memory attachment on top of Linux memory hotplug allows to save considerable effort by reusing existing code and proven technology, there is a main challenge associated with using it in the context of the proposed architecture. The hotplug mechanism needs to be well integrated with programmable logic reconfiguration, in a way that guarantees that physical addresses as seen by the operating system kernel and content of programmable logic hardware tables are consistent. Non Uniform Memory Access (NUMA) refers to a memory design for single-board multiprocessor systems where CPUs and memory modules are grouped in nodes. A NUMA node is a logical group of (up to) one CPU and the memory modules which are mounted on the board physically close (local) to the processor. Even though a processor can access the memory on any node of the NUMA system, accessing node-local memory grants significantly better performance in terms of latency and throughput, while performance of memory operations on other nodes depends on the distance of the two nodes involved, which, in traditional systems, reflects both the physical distance on the board and the architecture of the board-level memory interconnect between the processor and the memory module. When a new process is started on a processor, the default memory allocation allocates memory for that process from the local NUMA node. This is based on the assumption that the process will run on the local node and so all memory accesses should happen on the local node in order to avoid the lower latency nodes. This approach works well when dealing with small applications. However, large applications that require more processors or more memory than the local node has to offer will be allocated memory from non-local NUMA nodes. With time, memory allocations can become unbalanced, i.e., a process scheduled on a NUMA node could spend most of its memory access-time on non-local NUMA nodes. To mitigate this phenomenon, the Linux kernel implements a periodic NUMA balancing routine. NUMA balancing scans the address space of tasks and unmaps the translation to physical address space in order to trap future page faults. When handling a page fault, it detects if pages are properly placed or if they should be migrated to a node local to the CPU where the task is running. NUMA extensions of the Linux Kernel are exploited by our architecture as a means to represent remote memory modules as distinct NUMA nodes: we group remote memory chunks allocated to the brick into one or more CPU-less NUMA nodes. Each of these nodes has its own distance (latency) characterization, reflecting the different latency classes of remote memory allocations (e.g., tray-local, rack-local, or inter-rack). We develop a framework to dynamically provide the Linux Kernel with an overview of

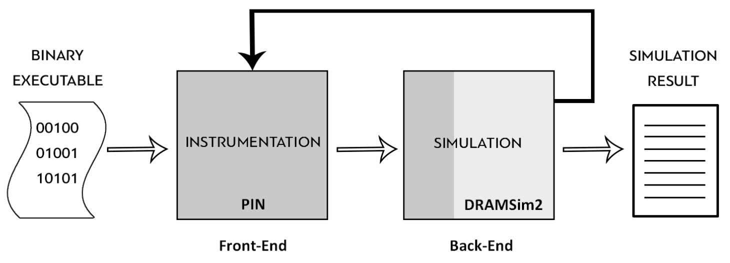

Fig. 11. A simplistic illustration of the DiMEM simulation tool for disaggregated memory.

the available memory every time new modules are hot plugged to the local system and also provide the distance (latency) between CPUs and allocated remote memory. This information facilitates the development and extension of current task placement and memory allocation techniques within the Linux Kernel for the effective scheduling of VMs to CPUs and for improved locality in memory allocation.

5 Simulating Disaggregated Memory

Disaggregated memory is of paramount importance to the approach of disaggregation in data centers, and the capacity to analyze and understand its effect on performance is a prerequisite for the efficient mapping of applications to such architectures. The current section describes a simulation tool for disaggregated memory, dubbed DiMEM Simulator [5, 19] (Disaggregated Memory Simulator), which facilitates the exploration of alternative execution scenarios of typical HPC and cloud applications on shared memory architectures, e.g., Raytrace and Data Caching.

DiMEM Simulator relies on Intel's PIN framework [16] for instrumentation and application analysis purposes, while DRAMSim2 [22] is employed for memory simulations. Figure 11 depicts a simplistic view of DiMEM Simulator. On the front end, DiMEM Simulator conducts functional simulations of the TLB and the cache hierarchy, and generates a trace file of accesses to main memory. The *Allcache* pin tool (developed by Artur Klauser [16]) for functional simulations of instruction/data TLB and the cache hierarchy served as the basis in order to support multithreading and hyper threading. Instrumentation is conducted at the instruction granularity level in order to detect memory accesses that are then analyzed to distinguish between cache hits and misses. The aim is to determine actual accesses to main memory, which inevitably follow the LLC misses. DiMEM Simulator maintains a list of records for every access to main memory, which is used to generate the required memory trace file.

On the back end, DRAMSim2 is deployed to analyze the generated memory trace file. A trace file example for DRAMSim2 is shown below, with the first column providing a virtual address, followed by the type of instruction and the cycle it was issued.

```
0x7f64768732d0 P_FETCH 1
0x7ffd16a5a538 P_MEM_WR 8
0x7f6476876a40 P_FETCH 12
0x7f6476a94e70 P_MEM_RD 61
0x7f6476a95000 P_MEM_RD 79
```

Once a certain amount of memory traces is generated, the so-called window, a penalty-based scoring scheme is employed to determine the cycle each instruction is accessing memory. Sorting instructions by the memory-access cycle facilitates simulations in multithreaded execution environments. Given the two types of memory that DiMEM is simulating, i.e., local and remote (disaggregated), two DRAMSim2 instances are employed. The second instance is required to model remote memory correctly, given the additional latency of the interconnect.

To evaluate DiMEM Simulator, a variety of HPC and cloud benchmarks were employed. HPC benchmarks involved Barnes, Volrend, and Raytrace of the Splash-3 benchmark suite [23], and FluidAnimate of the PARSEC benchmark suite [6]. The memcached-based Data Caching benchmark of the CloudSuite [11, 18] benchmark suite for cloud services was employed as the cloud benchmark. All benchmarks were evaluated based on a series of processor configurations that were modeled after modern, high-end microprocessor architectures. Four different memory allocation schemes were employed, with an increasing percentage of remote memory, i.e., 0%, 25%, 50%, and 75%, as well as varying latency for remote accesses, ranging between 500ns and 2,000ns.

Figures 12 and 13 illustrate Raytrace and Data Caching Workload profiles based on LLC misses measured in Misses per Kilo Instructions (MPKI) per window. As can be observed in the figures, the workload of the Raytrace application exhibits significant variations over time, whereas the Data Caching one remains largely constant. This directs sampling accordingly in order to avoid prohibitively large evaluation times.

Figure 14 illustrates the effect of disaggregated memory on application execution in terms of induced overhead/slowdown and varying effective bandwidth as disaggregated latency increases from 500ns and up to 2,000ns. A RISC-based CPU with 4GB of total memory is employed in all cases, while the percentage of remote memory in the system varies from 25% to 75%. As expected, increasing disaggregated latency reduces the effective bandwidth of the remote memory equivalently in all three remote-memory-usage scenarios, i.e., 25% (Fig. 14b), 50%(Fig. 14d), and 75%(Fig. 14f). As the percentage of remote memory in the system increases, the negative effect of increased remote memory latency on application performance intensifies, as observed in Fig. 14a, Fig. 14c, and Fig. 14e. As can be observed in the figures, the increased access latency to remote memory reduces overall performance. However, the results suggest that maintaining

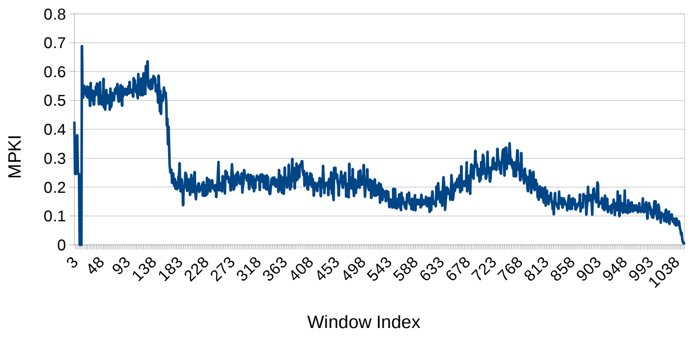

Fig. 12. Workload profile for Raytrace (Splash-3 benchmark suite [23]), measured as the number of missed per kilo instruction (MPKI) per window.

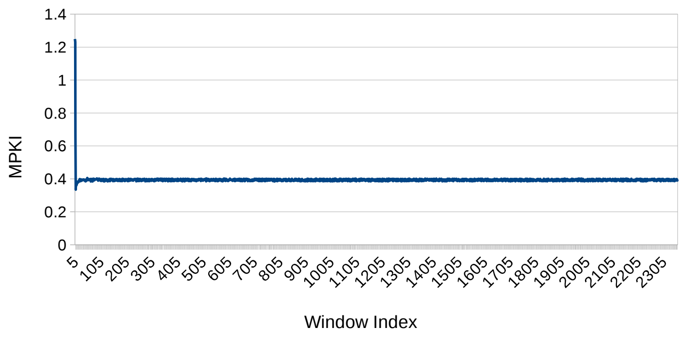

Fig. 13. Workload profile for Data Caching (CloudSuite benchmark suite [18]), measured as the number of missed per kilo instruction (MPKI) per window.

latency to disaggregated memory below 1,000 ns yields acceptable performance deterioration, allowing to benefit from the overall disaggregated approach.

6 Conclusion

The dReDBox project aims at achieving fine-grained resource disaggregation, with function blocks representing the basic units for creating virtual machines. This can lead to fully configurable data center boxes that exhibit the capacity to better serve application requirements, by quantitatively adapting resources to application workload profiles. Compute-intensive applications, for instance, will induce the allocation of increased amounts of CPU nodes, whereas memory-intensive applications will trade processing power for memory and storage resources. Evidently, the success of this approach relies on eliminating disaggregation-induced overheads at all levels of the system design process, in order to virtually reduce the physical distance among resources in terms of latency and bandwidth.

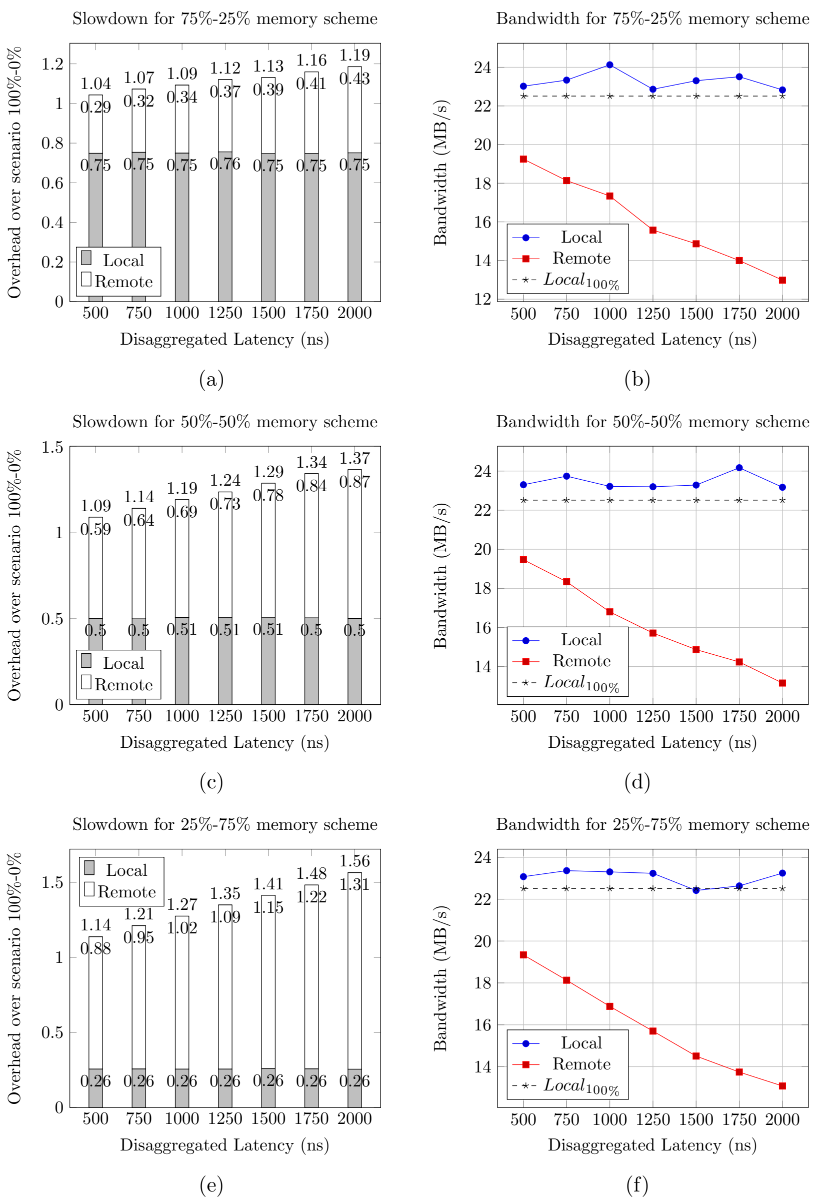

Fig. 14. The effect of disaggregated memory on application execution in terms of overhead/slowdown and effective memory bandwidth for varying disaggregated latency values.

Acknowledgements

This work has been supported by EU H2020 ICT project dRedBox, contract #687632.

Bibliography

[1] (2017) JanusGraph: Distributed graph database. `http://janusgraph.org/`

[2] (2017) OpenStack. `https://www.openstack.org/`

[3] (2017) OpenStack Horizon. `https://docs.openstack.org/developer/horizon/`

[4] (2017) OpenStack Nova. `https://docs.openstack.org/developer/nova/`

[5] Andronikakis A (2017) Memory system evaluation for disaggregated cloud data centers

[6] Bienia C, Kumar S, Singh JP, Li K (2008) The parsec benchmark suite: Characterization and architectural implications. In: Proceedings of the 17th international conference on Parallel architectures and compilation techniques, ACM, pp 72–81

[7] Caulfield AM, Chung ES, Putnam A, Angepat H, Fowers J, Haselman M, Heil S, Humphrey M, Kaur P, Kim JY, et al (2016) A cloud-scale acceleration architecture. In: Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on, IEEE, pp 1–13

[8] Chen F, Shan Y, Zhang Y, Wang Y, Franke H, Chang X, Wang K (2014) Enabling FPGAs in the cloud. In: Proceedings of the 11th ACM Conference on Computing Frontiers, ACM, p 3

[9] Dragojević A, Narayanan D, Hodson O, Castro M (2014) Farm: Fast remote memory. In: Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, pp 401–414

[10] Fahmy SA, Vipin K, Shreejith S (2015) Virtualized fpga accelerators for efficient cloud computing. In: Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on, IEEE, pp 430–435

[11] Ferdman M, Adileh A, Kocberber O, Volos S, Alisafaee M, Jevdjic D, Kaynak C, Popescu AD, Ailamaki A, Falsafi B (2012) Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: ACM SIGPLAN Notices, ACM, vol 47, pp 37–48

[12] Kachris C, Soudris D, Gaydadjiev G, Nguyen HN, Nikolopoulos DS, Bilas A, Morgan N, Strydis C, Tsalidis C, Balafas J, et al (2016) The vineyard approach: Versatile, integrated, accelerator-based, heterogeneous data centres. In: International Symposium on Applied Reconfigurable Computing, Springer, pp 3–13

[13] Klimovic A, Kozyrakis C, Thereska E, John B, Kumar S (2016) Flash storage disaggregation. In: Proceedings of the Eleventh European Conference on Computer Systems, ACM, p 29

[14] Lim K, Chang J, Mudge T, Ranganathan P, Reinhardt SK, Wenisch TF (2009) Disaggregated memory for expansion and sharing in blade servers. In: ACM SIGARCH Computer Architecture News, ACM, vol 37, pp 267–278

[15] Lim K, Turner Y, Santos JR, AuYoung A, Chang J, Ranganathan P, Wenisch TF (2012) System-level implications of disaggregated memory. In: High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on, IEEE, pp 1–12

[16] Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G, Wallace S, Reddi VJ, Hazelwood K (2005) Pin: building customized program analysis tools with dynamic instrumentation. In: Acm sigplan notices, ACM, vol 40, pp 190–200

[17] Mavroidis I, Papaefstathiou I, Lavagno L, Nikolopoulos DS, Koch D, Goodacre J, Sourdis I, Papaefstathiou V, Coppola M, Palomino M (2016) Ecoscale: Reconfigurable computing and runtime system for future exascale systems. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016, IEEE, pp 696–701

[18] Palit T, Shen Y, Ferdman M (2016) Demystifying cloud benchmarking. In: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp 122–132

[19] Papadakis O (2017) Memory system evaluation of disaggregated high performance parallel systems

[20] Pugsley SH, Jestes J, Balasubramonian R, Srinivasan V, Buyuktosunoglu A, Davis A, Li F (2014) Comparing implementations of near-data computing with in-memory mapreduce workloads. IEEE Micro 34(4):44–52

[21] Putnam A, Caulfield AM, Chung ES, Chiou D, Constantinides K, Demme J, Esmaeilzadeh H, Fowers J, Gopal GP, Gray J, et al (2014) A reconfigurable fabric for accelerating large-scale datacenter services. In: Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on, IEEE, pp 13–24

[22] Rosenfeld P, Cooper-Balis E, Jacob B (2011) Dramsim2: A cycle accurate memory system simulator. IEEE Computer Architecture Letters 10(1):16–19

[23] Sakalis C, Leonardsson C, Kaxiras S, Ros A (2016) Splash-3: A properly synchronized benchmark suite for contemporary research. In: Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on, IEEE, pp 101–111

[24] Tu CC, Lee Ct, Chiueh Tc (2014) Marlin: A memory-based rack area network. In: Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, ACM, pp 125–136

[25] Vipin K, Fahmy SA (2014) Dyract: A partial reconfiguration enabled accelerator and test platform. In: Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, IEEE, pp 1–7