Survey on memory and devices disaggregation solutions for HPC systems

Maciej Bielski,
Christian Pinto,
Daniel Raho
Virtual Open Systems
{*m.bielski, c.pinto, s.raho*}
@virtualopensystems.com

Renaud Pacalet
Institut Mines-Télécom,
Télécom ParisTech,
CNRS LTCI
pacalet@telecom-paristech.fr

***Abstract*—**

Traditionally, HPC workloads are characterized by different requirements in CPU and memory resources, which in addition vary over time in unpredictable manner. For this reason, HPC system designs, assuming physical co-location of CPU and memory on a single motherboard, strongly limit scalability, while leading to inefficient resources over-provisioning. Also, peripherals available in the system need to be globally accessible to allow optimal usage. In this context, modern HPC designs tend to support disaggregated memory, compute nodes, remote peripherals and hardware extensions to support virtualization techniques. In this paper, a qualitative survey on different attempts of memory and devices disaggregation is conducted. In addition, alternative future directions for devices disaggregation are proposed in the context of the work planned in the H2020 *dRedBox* project.

***Keywords*-HPC, memory disaggregation, cpu disaggregation, HPC virtualization**

I. Introduction

This paper is about a state-of-the-art study on most recent solutions to handle memory and devices sharing in High Performance Computing (HPC) systems, at the best of authors knowledge. Different methods that can contribute to the creation of modern system designs for HPC are compared in qualitative terms; this comparison survey is done from a virtualization support perspective since it makes resource management more flexible, better isolates different workloads and provides good fault-tolerance mechanisms.

According to current HPC deployments observations, the demand in memory grows much faster over usage time than for CPU [1]. The most often used approach taken by HPC system designers in order to cope with workloads demand is over-provisioning [2]. That means, that since CPU and memory are physically coupled together, they can only be upgraded proportionally, keeping the CPU-to-memory ratio constant. In this way, the most stringent resource requirement is met, while resources in excess are destined to not be fully exploited. The only exception is when a motherboard offers vacant DIMM slots, into which additional modules can be plugged, extending overall system memory, but this is only one-time improvement, not a scalable solution. Therefore, physical resources' co-location yields several drawbacks as described herewith.

A. Aggregation drawbacks

In case of CPU and memory, a need of upgrading one of these two components, would practically result in increasing the number of computing nodes. Then, additional nodes installed on the system take more space and raise the overall energy consumption with extra heating. It is important to keep in mind that HPC systems usually consist of large number of machines and therefore all this inefficiencies are exacerbated by the scale factor. Also, investing in resources that cannot be efficiently exploited by the system makes the investment far from optimal, to say the least, with poor Total Cost of Ownership (TCO), which, in turn, will penalize potential customers using the system.

Other than memory provisioning, an optimal design of an HPC system should consider to easy the access to peripherals. Some of them are intended for accelerating particular types of computing operations (like GPGPU) and may not be required to be attached permanently to all machines. However, it has been noticed, that in general it is good to have few of them globally accessible to accelerate some workloads, thus saving CPU clocks [3]. Another type of devices are network interface cards (NIC), which are in general widely used but their performance is critical because it has a big impact on application speed. Therefore, in data center systems, there are usually expensive high-end models deployed, which should also be shared across the system in order to amortize the cost and maximize utilization.

B. Desired solution characteristics

To overcome the aforementioned drawbacks, a novel HPC system architecture should be capable to satisfy the following characteristics:

- CPU and memory should be physically disaggregated to permit more flexible resources allocation, lower power consumption and easier maintenance. Related to this and for the purpose of this paper the notions of compute node and memory node will be used.
- Multiple nodes should be able to cooperate in a heterogeneous manner to allocate and use memory from a global pool in a dynamic way, according to application requirements.
- Such a novel system design should support virtualization techniques which are necessary to achieve together flexibility, good performance, security and fault-recovery.

- In order to keep the overall cost low, in comparison with custom solutions, the design should make as far as possible use of commodity components, while expensive devices should be shared to minimize their cost per compute node.
- Finally, the novel solution has to provide at least the same performance per dollar as state-of-the-art solutions.

C. Contribution

Designing a system meeting all these requirements is a challenging task. In the following sections, there are presented solutions attempting to achieve that, at least in some aspects. The main contribution of this paper is a juxtaposition and comparative analysis of existing concepts divided into two main categories: memory disaggregation and devices disaggregation and sharing.

D. Document organization

This work is structured as follows. Section II gathers different methods of memory disaggregation. Section III compares them. Similarly, section IV discusses several methods of devices disaggregation, which are then compared in section V. Section VI concludes the whole work and proposes further directions.

II. Memory disaggregation

This section presents various approaches for memory disaggregation from CPU nodes, based on the state-of-the-art analysis at the best of authors knowledge. The approaches surveyed can be differentiated in terms of: remote memory interconnection and access technique, location with respect to CPU (whether it is placed in a remote node or not), scalability (ready for providing more physical memory in future) and virtualization support (hypervisor modification). In the rest of the section all solutions will be first briefly described, and then compared according to various criteria.

A.

Velegrakis [4] presents a discrete prototype, consisting of two Avnet Zedboards[1] connected with an FMC cable. The main goal of the solution is to enable each CPU to access the memory of the remote nodes (the CPUs in other boards), so to create a bigger system memory abstraction that is not limited to the memory locally available for each CPU. Three main remote memory access techniques are proposed, based on a static physical addresses mapping performed by an FPGA block. Another FPGA block called *Chip2Chip* handles interconnection logic using AXI messages. In the first approach, direct access can be performed through local CPU cache as what is done for a regular memory access. Alternatively the local cache can be bypassed and the memory access is served through the remote CPU cache. This access method is completely transparent from the point of view of the host operating system.

[1]Based on Xilinx Zynq cores that embed an ARM CPU and an FPGA fabric in the same chip.

In a second approach, the remote memory can be accessed as an I/O device and configured as OS swap partition. Thus automatically used by the operating system to swap memory pages in case of high memory pressure. In the third approach, the remote memory can be seen as a character device and exposed directly to user space. Remote interrupts are provided between boards using the mailbox mechanism. The nice aspects of this work are that it proposes an heterogeneous solution on a widely used ARM processor, and that the direct access method is completely transparent for the OS.

The main drawback is that these prototype uses a custom protocol and relatively slow physical connection. Therefore its scalability is quite doubtful, and even more, it is questionable whether it allows to extrapolate relevant measurements in order to estimate the expected overhead of a real system.

B.

Lim et al. [5] present a Xen-based software prototype that models a disaggregated, non-heterogeneous system with one compute node and a remote memory blade connected through PCI Express (PCIe). Remote memory address mapping and protocol communication is handled by a specific component at memory node. On the computing node side, the hypervisor manages one Nested Page Table (NPT) per guest virtual machine and NPT entries are provided to the guest as its physical addresses. Nevertheless, some NPT entries have no corresponding local memory assigned (such technique is called VM memory overcommitment) and they are called remote. Each time a guest OS accesses an address that maps to a NPT remote entry, it causes a page fault in the hypervisor. On that event, the hypervisor has to swap-in a page from remote memory node to local memory and update the NPT table before passing the control back to the guest. In case there is not enough room in local memory some pages may have to be swapped-out beforehand. From guest's perspective local and remote memory is perceived alike, with the only difference that remote page access latency is much higher because of the additional operations required. The remote memory assignment is performed dynamically on page fault. This is a positive feature because usually guest OS memory utilization is maximum only in short peak moments while it is lower in average. Therefore, VM can be provided with less memory at startup and additional resources will be assigned only when required, with the possibility to be reclaimed once no longer required. From global perspective this solution allows for more dense memory utilization. The basis of this concept is that it is very unlikely that all virtual machines will reach their peak utilization at the same moment, and proper balancing allows to provide more virtual memory than there are physical resources actually available. Remote memory has however the drawback of a higher page fetch delay. It is worth noting that this is only a software simulation performed on a single machine, part of its memory has been called remote and its access delay is enforced in software. This design has following drawbacks. It is the hypervisor that makes a global-scale decision about which pages will be evicted.

Therefore, there are cross-VM interferences occurring when page faults caused by one VM can result in another VM's page eviction. Moreover, the swapping algorithm cannot make use of guest page information since they are held by guest OS structures.

C.

In [1] Lim et al. follow-up the previous concept (Sec. II-B) and create a real prototype, which consists of a global remote memory blade connected with several compute nodes. The architecture prototyped is both disaggregated and heterogeneous. Each compute node runs a hypervisor that can use a memory ballooning[2] technique to allocate remote pages. Compared to the previous software concept there are several important differences. First, there are multiple compute nodes with hypervisors, each of them has its own "System Memory Address Space", where the local part is placed at the bottom. The remote part, on the other hand, is mapped to disjoint memory ranges at the global memory node. Moreover, except for the swapping method, the authors propose another approach to access the remote memory at cache-block granularity. That, however, requires a specific hardware support, that is a custom coherence filter chip, which redirects cache-fill requests from a CPU to the remote memory. The positive aspect of the presented solution is its transparency from hypervisor perspective, negative ones are a custom chip requirement and accumulating latency when each remote memory access would require remote node interaction.

D.

Dragojević et al. [6] introduce the "FaRM" system, a cluster solution in which all participating machines are connected by RDMA over Converged Ethernet (RoCE) [7] network and expose some part of their memory to the others. Exposed memory is divided into addressable chunks and an address table is kept in the internal Network Interface Card (NIC) memory. The size of this memory limits the number of table entries and also determines the size of a single chunk (2GB in the presented prototype). It is possible to have more addressable chunks but it degrades performance since page tables have to be fetched over a PCI bus. Remote memory access is based on NIC-to-NIC Remote Direct Memory Access (RDMA) transactions. It is worth mentioning that this is the first presented solution using a global memory manager layer, here called *memory allocator*, that exposes an API for the memory allocation and transfer operations. Therefore this solution requires a modified OS but offers dynamic remote memory provision. Nonetheless, the systems scalability is limited by both NIC internal memory size and also by some software mechanisms of NIC drivers. Component machines are heterogeneous but this system is not a real disaggregation example since both CPU and memory reside within a single node.

[2] Memory ballooning allows the hypervisor to provide virtual machines more memory than what physically available (over-commit)

E.

Montaner et al. [8] [9] describe the "MEMSCALE" architecture intended for cluster environments. In the prototype all computing nodes are aggregated in the same motherboard with custom HTX interconnect [10]. Each node keeps a part of its memory for an OS running on local CPU but remaining memory may be used by other nodes. The memory access is designed so that from user space perspective it is visible as a *logical memory region*, which can stay within one physical node or span more of them. Remote memory reservation is assisted by the operating system that can extend and shrink its memory a dynamic way. Therefore, it is again a form of remote memory manager, however it is distributed over all participating OSes. Once the size of a logical memory region is negotiated, further accesses are performed completely by the hardware, without OS intervention. This is possible due to the AMD HyperTransport [11] technology. Each time a remote memory operation is scheduled, a CPU transfers the request to a custom *remote memory controller*, which dispatches it to another remote memory controller but located at destination node and the latter one can access its local memory. The main drawbacks of this solution are its lack of heterogeneity (same motherboard), no real disaggregation and also its custom interconnect that may be a limitation in terms of scalability.

F.

Hou et al. leverage the PCIe SR-IOV[3] standard to create a heterogeneous system prototype [3] with resource sharing, in particular memory but also devices that will be discussed later in this paper. The system backplane is a PCIe switch, with one root node attached through a Transparent Bridge (TB) and four leaf nodes attached through Non-Transparent Bridges (NTB). The NTB chip offers several communication mechanisms: 1) it performs memory address translation between two address spaces according to mappings configured by the attached sides, 2) a doorbell register facilitates interrupts delivery from one side to the other and 3) a scratchpad register allows for both sides CPUs communication. The memory sharing mechanism comprises two steps. In the beginning, two sides negotiate remotely accessible regions and setup the mappings accordingly. Then, these regions are visible to the other side and may be accessed in two ways: either directly, with each request being relayed by the switch (by a custom driver hooked on the memory management module) or using a DMA engine in the NTB chip (remote memory emulated by a virtual block driver). This system has several advantages. Since the backplane interconnect protocol is the same as the system bus, there is no encapsulation performed at the node edge. Together with an optimized DMA mechanism it offers the performance up to almost 3Gbps when copying data between two nodes. In addition, memory may be requested on demand by only one node at a time (no simultaneous sharing). As drawbacks we may count the lack of remote memory caching, scalability

[3]PCI Express extension, Single-Root I/O virtualization

limited to the number of ports of the PCIe switch and the lack of memory and CPU disaggregation by SR-IOV design.

G.

Finally, Tu et al. [12] present the "Marlin" system, a PCIe solution, based on the previous one (Section II-F). The first difference is the fact that the prototype comprises only two leaf nodes, in addition to the root node. Also, memory sharing is handled differently: instead of negotiating the access to remote memory regions, the whole system memory is exposed as a single address space. The management host first maps the memory of each leaf node to its physical address space and then passes that global address space back to the nodes. Correct address space mappings are set up at NTB ports accordingly. Except for this differences, memory access methods are the same as well as pros-and-cons characteristics.

III. Comparison of memory access methods

This section compares the methods presented in the previous section and draws some conclusions upon them. The ID of each solution corresponds to subsection in which it has been described.

Table I puts together all remote memory techniques with their characteristics. Columns 2 and 3 present different access methods. Cache-line granularity requires specific hardware support (e.g. AMD HT with remote memory controller) and may result in worse system performance if each access would require reaching remote node. This is the case of PCI Express technology since it offers no caching. On the other hand, swapping-in the whole page to the local memory means more data to be transfered once, but then, as long as requested data stays within the same page, it is available with local access latency. Intuitively, the optimal system should balance the fetched memory volume with implied access latency.

In columns 4 and 5 all solutions are differentiated according to disaggregation and heterogeneity features. However all positions present interesting methods only 2.C meets both requirements. Especially the concept of remote memory decoupled from CPU is not easy to implement in a real prototype and very often the limitations are related to the interconnect technology.

Column 6 collates all concepts according to the chosen interconnection technology. PCI Express is the most popular approach but does not support remote memory caching. On the other hand, AMD HT offers interesting features of a custom memory controller that could serve as a cache as well but this solution forces CPU manufacturer whereas PCI Express is an independent standard.

Column 7 considers the transparency of each solution from the OS (or hypervisor) perspective. Keeping in mind that II-A is a very specific *point-to-point* solution, it is visible that virtually every memory sharing mechanism requires some OS modification to be able to perform an operation like access negotiation, posting transaction or building global memory address space.

Finally, it would be appreciated to compare performance of all solutions. Unfortunately, there are independent works with different evaluation methods and, in most cases, there is no unified benchmarking technique. Nevertheless, sometimes it was possible to extract certain numbers. II-A reports high latency of 725 ns per single read/write operation (mostly due to 100 MHz FPGA clock) and a throughput of almost 0.9 Gbps (write) and above 0.3 Gbps (read) in direct access mode. With DMA, that uses several optimizations (e.g., AXI request interleaving), it is much higher, depending also on used CPU ports configuration (HP vs. ACP), between 2 and 5 Gbps. II-B, II-C and II-E do not mention relevant data for comparison. II-D claims to copy data between nodes 10 times faster comparing to TCP traffic using the same NIC cards and with at least 145 times lower latency. In II-F direct load operations offer low throughput of 0.2 Gbps because of serial processing, store performs better, around 2.5 Gbps. In DMA mode, in both directions the throughput grows together with block size, up to 3 Gbps. The last solution, II-G, achieves up to 20 Gbps of throughput and the time of double copy (back-and-forth) of 27 us. In general, DMA transfer mode offers superior performance comparing to direct copying.

A conceptual ideal solution would comprise an extensible set of remote memory nodes decoupled from multiple heterogeneous CPU nodes, with memory access granted to compute nodes by a management unit. At compute host level, guest OS would be provided with overcommitted page table and remote memory pages would be swapped-in or -out by hypervisor, possibly exploiting memory ballooning driver. In case of II-A the point-to-point interconnection extensibility is not known, memory and CPU are coupled together and the access is performed in direct manner. II-B assumes a scalable remote memory swapped by hypervisor and address mapping managed by separate system component but it proposes only one compute node. Solution II-C is disaggregated, heterogeneous and exploits swapping technique, however direct access method is also discussed. Its scalability is dependent on PCIe high-speed interconnect capabilities. II-D proposes remote memory access through NIC-to-NIC RDMA mechanism managed by memory allocator and specific API. The mechanism itself has limited scalability and CPU and memory are coupled together. In II-E, remote memory is accessed on a cache block granularity an introduced with transaction phase done by software. System nodes work heterogeneously but there is no CPU and memory disaggregation and also the interconnection is a custom one of a not-known scalability.

IV. Peripheral disaggregation

As already mentioned, another important trait of a disaggregated HPC system is a fair access to all peripherals (e.g., GPUs), that are installed on specific remote nodes. Usually devices are interfaced through registers mapped onto memory regions, therefore performing remote device operations is closely related to memory sharing. Analogically to the memory part, this section presents several concepts

ID	Direct access	Swap device	Heterogeneity	Disaggregation	Interconnect	OS visibility	Comments
II-A	X	X	Y	N	Custom (AXI-compatible)	YES, valid physical memory	Discrete prototype
II-B		X	N	Y	PCI Express	NO, swap on page fault	Software simulation
II-C	X	X	Y	Y		Block access - YES, swapping - NO	AMD HT
II-D	NIC-to-NIC		Y	N	Ethernet	NO, transactional access	Interconnected Ethernet cards
II-E	X		Y	N	AMD HT + HTX (custom)	NO, OSes negotiate	Same motherboard
II-F	X	X	Y	N	PCI Express	NO, modified drivers	
II-G	X		Y	N	PCI Express	NO, global address space	

Table I
REMOTE MEMORY TECHNIQUES COMPARISON

of systems with all installed peripherals accessible to each compute node. All solutions will be compared afterwards.

A.

The PCIe-based system by Hou et al. [3] (already mentioned in section II-F) presents also devices disaggregation method in two examples; GPGPU and NIC, correspondingly. The GPGPU is attached to its local node and exposed to rest of the system via NTB port. Posting a transaction by a compute node is divided in two phases. First one is negotiation of access to the region of memory, on which device ports are mapped. Once granted, in the second phase data are transferred to the device through DMA engine and processing is launched afterwards. Then, completion flag register is periodically polled and another DMA transfer fetches the output data once processing finish is detected. With such communication scheme, DMA-based remote device access is based on the same design principles as remote memory access over an NTB port. Other than GPGPU, next example presents NIC sharing concept. Each physical card has an associated virtual NIC (VNIC), emulated by a driver, with its own MAC, IP address and routing table. Additionally, the IP layer has been implemented over the PCIe link such that a remote NIC access is based on IP packets transfer, again performed by DMA under the hood. Both examples are based on DMA mechanisms and simultaneous sharing is automatically achieved, however at different granularity. A GPGPU can start next operation only once the previous one finished. Contrarily, different transfers can be multiplexed by NIC into one stream thanks to packet switching paradigm. Such system has two main drawbacks; its scalability is physically limited by the number of switch ports and also the VNIC emulation introduces an overhead that reduces the performance.

B.

Xu et al. propose a redundant system [13] based on PLX PEX 8796 [14] PCIe MR-IOV switch, equipped with a management CPU dispatching configuration transactions between attached hosts and devices connected by NTB ports. Unlike the SR-IOV, here all switch endpoints (attached devices) can be shared by multiple (up to four) different hosts and hosts can also communicate with each other. In this particular case there are two, primary and backup hosts. The former sends heartbeat messages whereas the latter traces its execution, performs periodic checkpoints and resumes from the last one on failover. There are different configurations possible where more hosts can be active simultaneously. This is a very interesting work, since all hosts could potentially run heterogeneous hypervisors that get access to physical functions (PFs) of attached devices without any custom hardware. Having that, each guest OS could exclusively access corresponding virtual function (VF) in direct pass-through manner, for example using VFIO drivers [15]. With SR-IOV, such solution would be limited to a single hypervisor only. Additionally, fault detection mechanisms offered by the switch could be very beneficial support for VM checkpointing and migration performed at hypervisor level. Similarly to all switch-based methods, capability of the switch is a scaling limitation. It is worth to emphasize, that, up to authors best knowledge, this is the first prototype that uses MR-IOV switch.

C.

Suzuki et al. described a prototype system [16] that attempts to reach similar functionality of sharing PCIe endpoints (devices) amongst multiple hosts. A virtual, distributed MR-IOV PCIe switch has been constructed from two types of custom adapters (downstream bridge - attached to a device and upstream bridge - attached to a host) attached to Ethernet network. Adapters perform tunneling PCIe packets over Ethernet. Each downstream bridge is identified by VLAN number and a separate system manager host reconfigures the network for proper device assignment. With one VLAN number per bridge all system peripherals can be shared only by taking turns (reconfiguration required on each access). In other variant, the SR-IOV compliant devices can be shared simultaneously. Then, downstream

bridge manages device's PF and assigns each upstream host with single VF. Therefore, if a host runs a hypervisor, this VF cannot be shared amongst guests without additional software emulation. This concept endeavors to solve the problem of sharing attached PCIe devices amongst heterogeneous upstream hosts in a scalable way, however it requires additional encapsulation and uses Ethernet interconnection. Both features heavily affect the latency.

D.

At the scope of a single machine, device access sharing can be realized in software. Garzarella et al. implemented the *ptnetmap* framework [17] that virtualizes NIC interface to enable network I/O at a high packer rates. It is based on the *netmap* framework, already used by QEMU [18] and KVM [19]. Assuming devices' SR-IOV compliance, instead of VF being directly accessed by guest OS, the hypervisor opens a netmap port associated with that VF and exposes it to the guest. One VF can have multiple associated netmap ports and assigned to different guests, therefore providing shareability. The design is characterized by high efficiency thanks to zero-copy approach. Device access control phase is handled the hypervisor but after that data is conveyed directly between guest and the device VF without hypervisor intervention. This solution requires modification of both host and guest OS. Moreover, this is one compute node perspective but it may be combined with other PCIe solutions that offer an assignment of a single VF per host.

E.

The *Ladon* architecture [20], by Tu et al. , is a prototype that shares PCIe SR-IOV compliant devices with multiple compute hosts in different PCIe domains. This is yet another endeavor to offer MR-IOV functionality without an actual MR-IOV switch. Memory requests can be exchanged trough interconnecting two-port NTB that performs necessary address translation and isolates attached domains, so that each one perceives the other as a single PCIe device. All devices are attached to management hosts in the *master domain* that handles NTB ports configuration. Compute hosts reside in *slave domains* and run hypervisors with multiple VMs. The BIOS at each compute host discovers PCIe devices allocated at NTB port and each time when a VM boots up, the enumeration queries are intercepted by the hypervisor (modified KVM), which passes its own configuration to the VM and installs a set of *virtual devices*. With such configuration, when VM wants to access the device, the hypervisor requests management hosts for binding an *actual device* with *virtual* one at NTB port. That way, a VM gets a direct access to an *actual device*, which could be a VF or PF of a *physical device*. Data transfer is performed by DMA transactions and interrupts propagation. Before a transfer, source and destination addresses translation is performed with hypervisor assistance. To avoid unnecessary copying, the system implements zero-copy optimization that improved the overall performance. This solution is based on standard hardware components, and modified hypervisor and management hosts' OS. It offers a direct device access for a VM and simultaneous device sharing - as long as management host binds an *actual device* with VF, since PF cannot be shared in SR-IOV scheme. There are two main drawbacks; firstly, all devices have to be assigned to the same *master domain*, which is scalability limitation. Additionally, the management host in that configuration is a single point of failure.

ID	Visibility	Sharing
IV-A	a) VNIC per VM b) GPGPU via DMA	IP over PCIe Separate DMA transfers
IV-B	PF per host	Dispatching by management CPU
IV-C	VF per VM VF per host	Taking turns Simultaneous
IV-D	Virtual port per VM	Simultaneous
IV-E IV-F	Actual device per VM	VF/PF binding by MH

Table II
DEVICE VISIBILITY AND SHARING SCHEME

F.

A follow-up of previous systemIV-E has been presented by the same authors under the name *Marlin* [12]. It is intended especially for NIC sharing in disaggregated racks. The system architecture is almost the same like its predecessor, with one significant difference that DMA transfer configuration is done completely by hardware so that transactions between two machines within the same rack are performed without any software intervention. The only required software modification is providing proper drivers. Moreover, the switch hosts Ethernet cards for inter-rack communication, which improves system scalability, however this interconnect implies significantly lower throughput in comparison with intra-rack communication.

V. COMPARISON OF DEVICES DISAGGREGATION METHODS

This section indicates most important similarities and differences between presented methods and some conclusions has been drawn at the end.

Almost all presented works focused on NIC sharing, except for IV-A, which also described the case of a GPGPU. This is probably because accessing a remote NIC with satisfying performance exposes great challenges, while NIC is a basic component of a computing system of almost any type and for sure in case of HPC sector. Usually, applications require NICs to offer fast transfer and small latency. In case of other devices, like GPGPU, the workflow is different, the input is provided, operation is scheduled and requesting node has to be notified upon completion. A common point of all approaches is using PCI Express for attaching a device to switch port or a custom hardware adapter.

At first, in table II all methods are differentiated according to how a device is perceived from VM or host perspective and how sharing has been achieved. Some methods proposed two variants, they are presented as a) and b) accordingly. Native PCIe-supported sharing mechanisms

ID	Limitation
IV-A	NO, ethernet interconnect
IV-B	YES, nb of ports
IV-C	NO, ethernet interconnect
IV-D	NO, software virtualization
IV-E IV-F	YES, nb of ports

Table III
SCALABILITY LIMITATIONS

(IV-B,IV-C,IV-E,IV-F) are obviously the fastest, however in that case all devices have to be always connected to the same switch. From virtualization perspective, in a pure SR-IOV system, guest OS can directly access device's VF, however that implies no simultaneous sharing between compute nodes since corresponding PF is assigned to a single specific one at each time. This can be mitigated by high performance software emulation (IV-D) that would require only one VF per hypervisor.

Table III compares all solutions with scalability as a criterion. The biggest constraint shared by all the techniques is the number of devices that may be attached to a single PCIe switch. The problem of management host being single point of failure may be mitigated by using MR-IOV switch, however according to authors' best knowledge, there are not many of them available on the market.

Most of concepts in this section presents some performance measurements of accessing remote NICs and the results are as follows. IV-A uses 1Gb NIC and achieves 85% of peak bandwidth and latency less than 2x higher when using the card remotely, comparing to the performance when it is locally attached. IV-B does not provide any performance evaluation. IV-C shows a prototype with shared 10Gb NIC and reports the card bandwidth (not a throughput) of 9.9 Gbps in the best case. IV-D is not meaningful for that comparison since it does not discuss disaggregated devices. IV-E and IV-F both use 10Gb NICs, achieving up to 9 Gbps throughput between two VMs on a compute host and management host, with zero-copy optimization (IV-E). The throughput falls down with smaller message size.

In theory, an ideal solution would assume all connected devices simultaneously shared by multiple compute nodes in a MR-IOV manner. Provided devices compliance, each hypervisor would have its PF assigned and corresponding VFs exposed to multiple virtual machines. IV-A provides simultaneous access by SR-IOV switch (scalability limit) and software-implemented IP layer, that affects its performance. IV-B is the only case of MR-IOV system but the switch has a fixed number of ports and it cannot be extended further. IV-C virtualizes PCIe switch by custom adapters attached to the backplane (Ethernet). The concept is very promising, however the backplane is far too slow for such application. IV-E and IV-F propose a way how to attach more hosts and devices to single SR-IOV switch, which is an improvement although single switch dependency still persist. IV-D is an example of efficient software emulation that can efficiently support any solution offering only single VF per hypervisor. In all cases, the technological limits presence enforces different trade-offs and usually better efficiency is gained for the cost of scalability or to the opposite.

VI. CONCLUSION

This paper starts introducing the motivations that drive towards the definition of a novel disaggregated HPC hardware design. Based on the previous assumptions the document describes and compares the most relevant techniques for memory and peripherals disaggregation, at the best of authors' knowledge. Such analysis presents current technology trends, indicates the limitations of current solutions, to help clarify some of the needs of future HPC systems and thus drive future research work.

PCIe-based remote memory systems are usually based on VM overcommitment and swapping-in a remote page on a page fault, supported by DMA engine. The AMD HT technology enables accessing remote memory at cache-block granularity but this requires a custom remote memory controller, which, in theory, could also serve as a cache. Finer-grained access implies smaller latency but accumulating faster, with each access.

For recent systems PCIe interconnection seems to be a standard. Ethernet has not been designed for high-speed backplane purpose and cannot really serve as a primary solution for that purpose. PCI SR-IOV compliant devices offer good sharing mechanism but limited to a single compute host. Alternatively, larger number of compute nodes is possible with two-port NTB and proper memory regions translation. Ideally, MR-IOV could solve this problem but this standard is not well adopted yet, in particular it has been found only one solution based on commercially available product.

A. Further directions

This paragraph suggest promising solutions for a novel HPC system conception, inspired by the *dRedBox* H2020 EU project [21] [22]. It assumes disaggregated structure of multiple heterogeneous compute nodes and remote global memory nodes. Further works could consider PCIe technology as a primary interconnect and also emphasise virtualization hardware support in order to benefit from near-native performance while still keeping all virtualization merits. Remote memory management layer is mandatory so as to avoid data corruption but also to create an interface for deploying different access policies. As long as remote memory segments assigned to different compute nodes do not overlap, there is no need for any cache coherency between compute nodes which could limit its scalability as well as reduce the available bandwidth (i.e., cache coherency protocols generate messages that drain the bandwidth available for actual data). All system nodes and management host has to be connected to a common backplane that would virtualize a PCIe switch and introduce minimum possible latency, which is the most crucial constrain. This way system scalability is not limited to the number of ports of a specific switch, however it may require additional backplane adapters. Memory disaggregation may be achieved by VM overcommitment

and DMA-supported page swapping because this solution should work much faster than direct access on a cache line granularity and does not require a custom memory controller. Ideally, it would use memory ballooning technique for dynamic memory provisioning (i.e., claiming/returning additional memory resources to/from the global pool). This can be helpful to provide efficient load balancing for global memory. Device management and access can be efficiently implemented by para-virtualized drivers with front-end and back-end sharing common communication channel located in a remote memory. This requires an interaction with management host only for access negotiation but not during data transfer. An example of a solution that could be adapted is the 'virtio' drivers family [23].

The previous comments intend to identify the traits of the next generation HPC systems, providing at the same time high performance together with flexibility and much more efficient management with respect to current deployments.

ACKNOWLEDGMENT

This work was supported by the *dRedBoX* project [21] [22]. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 687632. This work reflects only authors' view and the EC is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] 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 *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 267–278.

[2] T. Patki, "The case for hardware overprovisioned supercomputers," pp. 36–38, 2015.

[3] R. Hou, T. Jiang, L. Zhang, P. Qi, J. Dong, H. Wang, X. Gu, and S. Zhang, "Cost effective data center servers," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 179–187.

[4] J. Velegrakis, "Operating system mechanisms for remote resource utilization in arm microservers , may 2015," *Technical Report FORTH-ICS / TR-452*, 2015.

[5] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 2012, pp. 1–12.

[6] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "Farm: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[7] H. Subramoni, P. Lai, M. Luo, and D. K. Panda, "Rdma over ethernet—a preliminary study," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–9.

[8] H. Montaner, F. Silla, H. Fröning, and J. Duato, "Memscale™: A scalable environment for databases," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. IEEE, 2011, pp. 339–346.

[9] H. Montaner, F. Silla, and J. Duato, "A practical way to extend shared memory support beyond a motherboard at low cost," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 155–166.

[10] H. Litz, H. Fröning, M. Thürmer, and U. Brüning, "An fpga based verification platform for hypertransport 3. x," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 631–634.

[11] AMD Inc., "Hypertransport ™technology: Simplifying system design," 2002.

[12] C.-C. Tu, C.-t. Lee, and T.-c. Chiueh, "Marlin: a memory-based rack area network," in *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2014, pp. 125–136.

[13] S. S.-D. Xu, C.-H. Wang, T.-C. Chang, and S.-F. Su, "Multi-root i/o virtualization based redundant systems," in *Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on*. IEEE, 2014, pp. 1302–1305.

[14] Pci express mr-iov switch, product overview. [Online]. Available: http://docs.avagotech.com/docs/12351860

[15] Vfio - "virtual function i/o". [Online]. Available: https://www.kernel.org/doc/Documentation/vfio.txt

[16] J. Suzuki, Y. Hidaka, J. Higuchi, T. Baba, N. Kami, and T. Yoshikawa, "Multi-root share of single-root i/o virtualization (sr-iov) compliant pci express device," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010, pp. 25–31.

[17] S. Garzarella, G. Lettieri, and L. Rizzo, "Virtual device passthrough for high speed vm networking," in *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*. IEEE, 2015, pp. 99–110.

[18] Qemu, system emulator and virtualizer. [Online]. Available: http://wiki.qemu.org/Main_Page

[19] Kvm, linux kernel-based virtual machine. [Online]. Available: http://www.linux-kvm.org/page/Main_Page

[20] C.-C. Tu, C.-t. Lee, and T.-c. Chiueh, "Secure i/o device sharing among virtual machines on multiple hosts," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 108–119, 2013.

[21] The dredbox project website. [Online]. Available: http://www.dredbox.eu/

[22] K. Katrinis, G. Zervas, D. Pnevmatikatos, D. Syrivelis, T. Alexoudi, D. Theodoropoulos, D. Raho, and C. Pinto, "On interconnecting and orchestrating components in disaggregated data centers: The dredbox project vision," p. to appear.

[23] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 95–103, 2008.