One Interface to Rule them All: A Hardware/Software Co-Design for Disaggregated Computing

Lluís Vilanova
Technion
vilanova@technion.ac.il

Yoav Etsion
Technion
yetsion@technion.ac.il

Mark Silberstein
Technion
mark@ee.technion.ac.il

1. INTRODUCTION

Datacenters are moving towards a paradigm of pooling resources (e.g., CPUs, storage and accelerators) into separate nodes to lower costs through easier hardware upgradability and higher resource utilization when running applications with heterogeneous demands.

A single request to an application can trigger a chain of accesses to multiple devices, but each device has wildly different hardware capabilities which expose vastly different data and control interfaces. As a result, applications cannot *securely* span all these devices in a way that keeps the cost and simplicity benefits of disaggregation while maintaining *efficiency*.

In this paper, we propose extending NICs to implement a model of *continuation-based computations* inspired in dataflow, which is used to weave the execution flow of applications across hardware devices without the need for each device to know each other's communication protocol. To achieve this, we lean on the observation that modern technology trends like device self-virtualization, multi-queue designs, RDMA and remote device transports (e.g., NVMe over fabric [14]) can be extended to allow devices to interact with each other without the need for intermediate software layers. Existing NICs can be easily extended to trigger such continuations as a response to device command completions, translating a continuation into a request directed at the next device on the processing pipeline.

2. PROBLEM STATEMENT

Resource disaggregation promises higher datacenter utilization and upgradability, ultimately leading to lower total cost of ownership (TCO). While the term was introduced in the context of decoupling memory and compute elements in a node [9], the same concept has already been applied to other types of resources like storage [14], FPGAs [16], GPUs [4] and TPUs [7]; we focus on this last case when using the term *disaggregated computing*. Nevertheless, there is no established theory in how to best operate such systems in the presence of complex applications that span multiple resources.

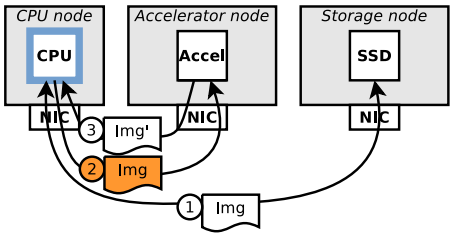
Let us take an example of an image store that performs simple real-time operations like image resizing and enhancement, as reported by Flickr [6]. The image store server performs a lookup in an in-memory cache for rapid request serving of cached, post-processed images. On a miss, it looks up the original image in storage (since storing all possible post-processed images requires too much space), reads it out, passes it to an image processing accelerator, caches the result, and sends it to the requesting client.

With the disaggregated computing paradigm, each resource (CPU server, storage and accelerator) resides on a separate node, from which it can be assigned to different applications on demand, and the whole rack (or datacenter) can be thought of as a machine that uses the network to interconnect all its devices.

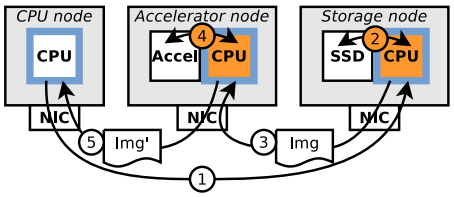
Unfortunately, existing deployments use general-purpose CPUs as either a central management point or as an executor of distributed application logic on each node (see Figure 1), leading to sub-optimal performance and higher TCO. In an ideal design, the image store server would instead send a storage read request, the storage device would directly place the image on the accelerator and trigger the appropriate processing operators, and the end of the accelerator processing would forward the enhanced result straight into the server and trigger it to respond to the original client.

In existing centralized designs (Figure 1a), the image server is used as a central point to manage all resources, where all data is always transferred between the server node and the storage or accelerator nodes. This provides very lean storage and accelerator designs (lower TCO), at the expense of increasing bandwidth requirements by 50% (we go from two to three image transfers; step (2)), and network latency by 33% (we go from three one-way messages to two round-trip messages; step (2)).

In existing designs that co-locate CPUs with every device (Figure 1b), applications can distribute their logic on each resource node to avoid the overheads of the centralized design. Nevertheless, CPUs are known to have problems scaling request processing to line-rate [3], co-locating a general-purpose CPU with each device increases the TCO (more so as CPU performance must scale with the performance of the co-located device), the OS on each CPU must isolate and multiplex logic from different applications, and programming the sys-



(a) Centralized logic design with device-aware NICs.



(b) Distributed logic design with CPU replication.

Figure 1: Examples of disaggregated designs and how they relate to resource access orchestration. Equipment and operation overheads have an orange background.

tem becomes 67% more complex: the application goes from three components (server, storage and accelerator) to five (for the two added CPUs on the storage and accelerator nodes; steps (2) and (4)).

2.1 Requirements

The thesis of this work is that disaggregated computing needs to provide resource composition, resource authorization and efficient request dispatching.

Resource composition is key to build systems that can perform complex operations in a decentralized and efficient way. The idealistic image server design above is a clear example: the storage node should place its result on the accelerator and immediately trigger the necessary image processing operations, while the accelerator should likewise send its result to and trigger a response from the image server.

We cannot simply make the hardware devices aware of each other's communication protocols to solve this problem; this is clearly unfeasible, especially as the number of hardware and software resources increases.

Resource authorization is key for system security. For example, the system cannot allow an application to read a storage block into the memory of another node that has not been previously assigned to the application.

This is not straightforward to achieve; access control must be globally orchestrated, and enforced at each physical device with knowledge of what application is every request being processed in lieu of.

Efficient interfaces are key to avoid unnecessary overheads. Using general-purpose CPUs to process the requests of each device is known to be difficult to scale in terms of throughput and latency [3]; hardware support is thus necessary to allow various resources and nodes to directly interact with each other.

2.2 Opportunities

Fortunately, some existing technologies are already pointing on the right direction, but we need to generalize their mechanisms to solve the larger problem space of the disaggregated computing paradigm.

Modern NICs offer virtual functions (VFs), so that applications can use the network without OS mediation [15]. NICs also offer multiple concurrent queue pairs, so that different application threads can use the network without synchronizing. Similarly, NVMe supports multiple queues and per-queue block addressing spaces, allowing applications to directly access local storage. Together, these features provide better throughput and latency without sacrificing security [1–3]. Some NICs also have RDMA capabilities, which improve throughput and latency of data transfers by bypassing the general-purpose CPU of remote nodes [13]. Similarly, NVMe over fabrics (NVMeOF) [14] specifies a protocol that uses RDMA to allow CPU servers to directly operate NVMe devices over the network. Some vendors provide programmable NICs [5, 11], which offer a cost-effective way to support NVMeOF: the target NIC manipulates a storage read request to point to the target node's local memory, detects when the request finishes, and performs an RDMA write of the result and response queue entry to the memory on the requesting CPU.

The network is thus a medium shared between different protection domains (different applications). Fortunately, existing network technologies like VLAN tags are used by NICs and other networking equipment to contain network traffic within a set of virtual network endpoints (i.e., a protection domain).

We can thus argue that existing hardware trends offer partial solutions to the problems raised by the disaggregated computing paradigm. What we still need to define, though, is a trusted interface that allows setting up all resources and communication channels for each application, together with a uniform mechanism that allow all devices (CPUs, storage and accelerators alike) to directly communicate with each other without mediation of a trusted layer of software.

3. SYSTEM DESIGN OVERVIEW

We propose the *Caladan*[1] system to solve the problems cited above. At its core, Caladan uses a homogeneous interface based on request/response queue pairs and a *continuation-based dataflow model* to execute complex datacenter-scale applications spanning multiple resources (i.e., in a composable manner) efficiently. This is a model based on hardware/software co-design; applications directly express resource requests and their relations, and programmable NICs are extended to en-

[1]Caladan is an ocean planet in the Dune universe. Its surface is predominantly covered with water, and its climate is characterized by much precipitation and strong winds, but is tolerable enough to make special and expensive weather control measures unnecessary.

capsulate, transport and trigger these requests and the resources they reference (similar to what NVMeOF-capable NICs already do).

Continuations are a well-known abstraction used to represent the state of a computation [17]. Many languages expose them as data structures that can be passed as function arguments, which in turn can invoke these arguments to *continue* the execution of said computation (e.g., event-based frameworks like NodeJS use them in the form of lambda callbacks). In Caladan, continuations are at the core of **resource composition**; they are sent as part of a resource request, and triggered by a remote NIC as a response to the completion signal of such request. Continuations encapsulate a regular resource request and can be nested within other continuations, so that computations across different resources can be composed into complex patterns without involving any additional software layers.

The result is an emergent **dataflow** model of computation [19]. A response is created only after a request's result has been generated, immediately triggering the computation represented by the continuation. Interestingly, this makes existing hardware and software designs indistinguishable: hardware devices will process a request when it has been published on a request queue, whereas software services will trigger a computation when they read a request from their network queue pair.

Finally, Caladan uses **object-capabilities** [12] to express the resources available to an application (a protection domain) in a secure way, providing **resource authorization**. Capabilities are protected handles that identify and authorize access to resources, which correspond to request/response queues in Caladan. Caladan's trusted computing base (TCB) is composed of a trusted OS that controls the creation of capabilities (the *control plane*), and the NIC and network infrastructure that controls how capabilities can be communicated and operated to access physical resources (the *data plane*).

A key observation is that Caladan can lean on existing datacenter-grade hardware that already supports RDMA, allowing efficient memory transfers to/from remote devices, and NVMeOF, allowing transport of request/response queue entries across the network in a secure way (i.e., an NVMeOF request cannot write into memory not available to the requesting application).

Caladan is designed around four core concepts, exemplified in Figure 2: (1) *virtual resources*, (2) *namespaces*, (3) the *invoke operation*, which accepts virtual resources as arguments, and (4) *continuations*, which can be used to chain requests across virtual resources.

Virtual Resources.

Virtual resources encapsulate any portion of a physical resource, like memory, virtual storage (i.e., protected through NVMe namespaces), a virtual accelerator (i.e., providing a small storage area for incoming requests and the ability to process them), or even a regular process. They are managed by the TCB, and references to them

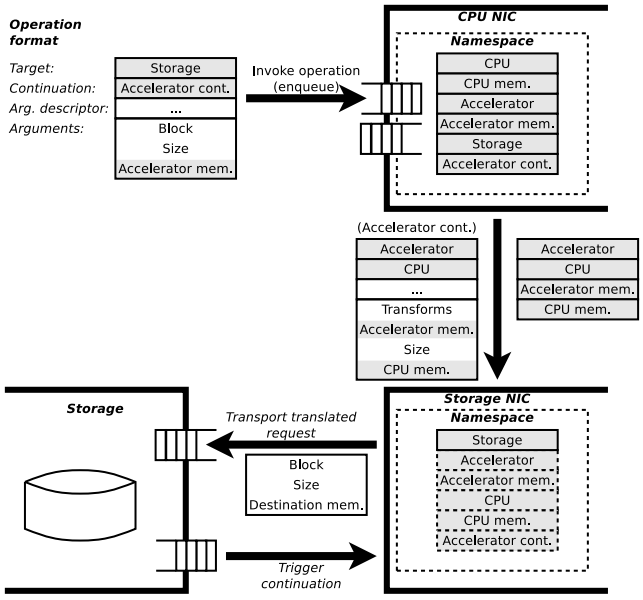


Figure 2: Example of the main elements in Caladan. Gray boxes represent references to object-capabilities available in a namespace.

are therefore capabilities.

Namespaces.

Namespaces are an interface to manage and authorize access to virtual resources. Every virtual resource exists on its own namespace, and on the namespace used to create that resource. Resources are referenced as an index into a namespace, can be transferred across namespaces as part of the arguments of a request to another virtual resource (thus preventing the confused deputy problem [8]), and are invalidated on the target namespace after a request has been responded to (providing a form of distributed and dynamic access control that solves the scalability problem of dynamic changes to translation and authorization structures [10, 21]).

Namespaces are implemented by the NIC as a table mapping virtual resources to the corresponding remote queue pairs of a target device, and are exposed as a regular hardware device through its own queue pairs.

The Invoke Operation.

Namespaces support a single invoke operation with a known format, allowing the NIC to inspect and forward it to the relevant resource. The invoke operation has a target resource, a continuation (see below), an argument description field, and a sequence of immediate and virtual resource arguments.

The NIC uses the virtual resource table to look up the destination queue of the target resource. The trusted OS is also a virtual resource, used for control plane operations (e.g., virtual resource creation).

The argument description field identifies which arguments are virtual resources, so that the NIC can prop-

erly translate them into values that the remote device can use. After translation, the NIC copies the arguments as an entry into the request queue of the remote device, and signals the arrival of a new request on it.

Arguments referencing memory are expressed as the memory's resource index and an offset into it, which the NIC translates into an address that a remote device can use for RDMA. This approach is similar to what NVMeOF-capable systems already do; a local request entry is copied into the request queue of the remote NVMe device, and the NICs use RDMA to transport the data between the device and the local memory buffer addresses set by the CPU on the request.

Continuations.

Continuations are created by invoking a special *continuation factory*, and its arguments represent an invoke operation (a target resource, continuation, etc.). This produces a new virtual resource that can be used as the continuation of a future invocation (a namespace can be used as a continuation to get the result of a request).

The contents of a continuation are transferred to the target NIC together with the request, but are not written into the target resource's queue. Later on, when the target resource signals a request completion (e.g., through an interrupt), the NIC captures that event and reifies the continuation as a new invoke operation. Note that capturing a device's request completion is already part of NVMeOF-capable NICs.

Since a continuation can have arbitrary contents, this can be used to trigger the next stage of a complex computation without having to know its interface.

3.1 Example Usage

Let us now look at how to implement the image store example in Caladan. When the server has a miss on its cache, it will prepare the image post-processing pipeline using continuations. Note that the steps are described in the application's logical order, but the continuations must be constructed on the reverse order (last to first; simple language support can be used to simplify this).

The server will lookup the storage location of the original image, and build a request for the storage resource (target resource) to read a certain offset and size (immediate arguments) into the accelerator's memory (virtual resource). This request is shown at the top of Figure 2, where the gray boxes indicate references to virtual resources located on the server's namespace. When enqueued, the operation and the resources it references are transferred into the storage's namespace (left side of Figure 2), and the NIC translates and copies the request into the target device request queue (bottom, similar to NVMeOF-capable NICs). The continuation field will be triggered when the NVMe device signals a response to the request (bottom).

The server will build the storage's continuation as a request to the accelerator (target resource) to read the original image from the accelerator's memory (virtual resource), with a given size and list of operators to apply (immediate arguments), and the image server's memory to write the post-processed image to (virtual resource). The continuation field will be triggered when the accelerator device signals a response to the request.

The server will then build the accelerator's continuation as a request to one of its namespace's queues (target resource) with a pointer to the client's request context (immediate argument). When the continuation is triggered, the server will use the client context pointer on the response message to resume the request, adding the post-processed image into its cache and sending it out to the client.

4. CONCLUSIONS

Caladan provides a *continuation-based dataflow* model to build complex large-scale applications under the disaggregated computing paradigm. It extends existing techniques like device self-virtualization, multiple queue pairs, RDMA and NVMeOF protocols to compose computations across unrelated devices without involving additional software layers. Caladan uses the concept of continuations as self-contained requests to other devices, such that a requesting device does not need to know the protocol of the next device on the computation pipeline. Given their simplicity and the maturity of existing technologies like programmable and NVMeOF-capable NICs, continuations and remote device operation can be efficiently implemented in modern NICs.

Works like LegoOS [18] and OmniX [20] describe solutions to very related problem spaces. LegoOS provides a distributed OS design for disaggregated memory systems, but does not concern itself with the composition of computations across resources. OmniX does look at the problem of near-data computation by composing application logic distributed across programmable hardware in a single node, but does not touch on the problem of lack of device programmability. Together, LegoOS and OmniX do not solve the problems covered by Caladan, but putting all three together could be a way to realize a full stack for hyper-efficient disaggregated infrastructures.

The design of Caladan still holds many open and interesting questions. It is still not clear whether support for stream-like transfers is necessary, for cases where the address or size of a piece of data is not known in advance. There is no definition of how to support pipelines with conditional forks, although one could envision using multiple continuations mapped to different request completion signals. It is not clear how to support remote operations that would generate various response signals, like getting one response after posting a kernel execution on a GPU and later receiving a response after the computation is finished. There is no clear way to define reusable computation pipelines, where an application can simply tweak some of the argument values to trigger a new computation. Nevertheless, the proposed

design is an interesting first step towards efficient, composable computations in disaggregated infrastructures.

References

[1] Data plane development kit (DPDK). https://www.dpdk.org/.

[2] Storage performance development kit (SPDK). https://spdk.io/.

[3] A. Belay, G. Prekas, , A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, Oct 2014.

[4] J. Duato, A. J. Peña, F. Silla, R. Mayo, and E. S. Quintana-Ortí. rCUDA: Reducing the number of gpu-based accelerators in high performance clusters. In *Intl. Conf. on High Performance Computing and Simulation (HPCS)*, Jun 2010.

[5] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg. Azure accelerated networking: SmartNICs in the public cloud. Apr. 2018.

[6] Flickr. A year without a byte, 2017. https://code.flickr.net/2017/01/05/a-year-without-a-byte/.

[7] Google. Cloud TPU. https://cloud.google.com/tpu/.

[8] N. hardy. The confused deputy (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, Oct 1988.

[9] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. In *Intl. Symp. on Computer Architecture (ISCA)*, Jun 2009.

[10] A. Markuze, A. Morrison, and D. Tsafrir. True IOMMU protection from DMA attacks: When copy is faster than zero copy. In *Intl. Conf. on Arch. Support for Programming Languages & Operating Systems (ASPLOS)*, Apr 2016.

[11] Mellanox. *BlueField Multicore System on Chip*, 2018.

[12] M. S. Miller. *Robust composition: towards a unified approach to access control and concurrency control*. PhD thesis, Johns Hopkins University, 2006.

[13] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot. The case for RackOut: Scalable data serving using rack-scale systems. In *ACM Symp. on Cloud Computing (SoCC)*, Oct 2016.

[14] NVM Express. *NVM Express over Fabrics, Revistion 1.0a*, Jul 2018.

[15] PCI-SIG. *Single Root I/O virtualization and sharing specification*, revision 1.1 edition, Jan. 2010.

[16] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, D. Burger, J. Larus, G. P. Gopal, and S. Pope. A reconfigurable fabric for accelerating large-scale datacenter services. In *Intl. Symp. on Computer Architecture (ISCA)*, Jun 2014.

[17] J. C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation - Special issue on continuationsâĂŤpart I*, Nov. 1993.

[18] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. LegoOS: A disaggregated, distributed os for hardware resource disaggregation. In *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, Oct 2018.

[19] R. M. Shapiro and H. Saint. The representation of algorithms as cyclic partial orderings. Technical report, NASA, July 1971.

[20] M. Silverstein. OmniX: an accelerator-centric OS for omni-programmable systems. In *Workshop on Hot Topics in Operating Systems (HotOS)*, May 2017.

[21] L. Vilanova, M. Ben-Yehuda, N. Navarro, Y. Etsion, and M. Valero. CODOMs: Protecting software with code-centric memory domains. In *Intl. Symp. on Computer Architecture (ISCA)*, Jun 2014.