Systems Software for Fast Inter-Machine Page Faults

Joel Nider
IBM Research, Haifa
joeln@il.ibm.com

Mike Rapoport
IBM Research, Haifa
rapoport@il.ibm.com

Yiftach Binjamini
IBM Systems, Haifa
yiftachb@il.ibm.com

***Abstract*—Cloud computing abstracts the underlying hardware details from the user. As long as the customer Service Level Agreements (SLA) are satisfied, cloud providers and operators are free to make infrastructural decisions to optimize business objectives, such as operational efficiency of cloud data centers. By adopting a holistic view of the data center and treating it as a single system, a cloud provider can migrate application components and virtual machines within the system according to policies such as load balancing and power consumption. We contribute to this vision by removing architectural barriers for workload migration and reducing the downtime of migrating processes. We combine the post-copy approach to workload migration with a novel specialized low latency interconnect for handling resulting remote page faults. In this work, we introduce a cross-architecture workload migration system, specify the requirements towards the specialized interconnect, discuss design trade-offs and issues, and present our proposed SW-HW co-design.**

***Keywords*—cloud; post-copy; migration; page fault; low latency; interconnect**

I. Introduction

We discuss the required characteristics of host software to support low-latency page faults for use during post-copy migration in a cloud data center. We enumerate the ideal attributes of a specialized interconnect for transferring memory pages between machines. Finally, we describe a prototype interconnect currently under design, why the prototype differs from the ideal, and the initial changes required in a specific operating system (Linux) to support the new interconnect. Since the solution relies on the virtual memory subsystem of the operating system, it can be applied to several different technologies that use the post-copy technique including virtual machines, containers, and processes. As such, we refer to all such technologies as *execution contexts* or contexts for short.

a) Context Migration: In the world of PaaS (Platform-as-a-Service) and SaaS (Software-as-a-Service) clouds, container migration can be used in the same ways that virtual machine migration is used in IaaS (Infrastructure-as-a-Service) clouds, and how process migration was used in past systems such as MOSIX [1]. These uses include load balancing, evacuation, optimization of system performance due to co-location of workloads, optimizing for power consumption [2] or migrating code to be closer to a data source to reduce network traffic. Container migration has additional use cases such as taking advantage of machines with particular hardware configurations for short phases of execution [3].

b) Downtime: Execution contexts are migrated by using a checkpoint-restore mechanism, which freezes the running context, dumps all state to a set of files which are copied to another machine for restoring [4]. The context is unresponsive from the time it is frozen until the time it is restored on the target machine [5], which is known as the *downtime* of the context. As shown in Fig. 1, the downtime is largely due to the dump and restore of the memory, and increases linearly with the size of the memory image [5], [6]. Minimizing the downtime of an execution context is often critical to the performance and responsiveness of a cloud application.

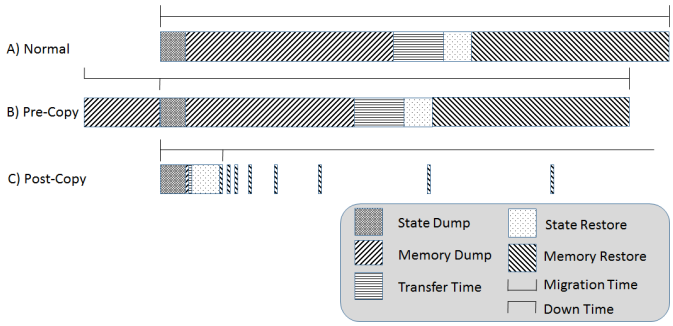

Fig. 1. Qualitative breakdown of time during checkpoint and restore of a single-threaded process.

c) Pre-copy: Downtime can be reduced by other methods such as pre-copy migration, as has been shown in virtual machine migration [7]. However, this technique has several drawbacks. First, the memory of a running application is constantly changing, and that implies that at least some memory pages will have to be copied multiple times (the page must be recopied every time it changes) which can increase the network traffic substantially. In order to know which pages have changed, the migration manager must track memory usage to identify which pages have changed since they were last copied. VMs may rely on a hypervisor for tracking memory changes since it has complete knowledge of guest memory accesses. However, for memory intensive applications such as databases, the set of changing pages may never converge, causing endless rounds of copying without ever completing the migration.

d) Post-copy: Post-copy migration drastically reduces downtime by postponing copying the memory until the target machine tries to access it [4]. This is more efficient than restoring the entire memory image up front for two reasons. Firstly, only pages that are actually required are copied, which can be considerably less than the total [6]. Secondly, the execution context can do an increasing amount of useful work between page faults, amortizing the time required to serve the faults over a longer period, effectively hiding the cost of the migration. There as some drawbacks as well. Even though only pages that are actually required are copied, each access requires a round-trip over the network to retrieve the page

during which time the context must wait. Also, if the source machine or network go down during migration the context may be lost completely as some essential memory may become unavailable.

e) Containers: Containers have been around for years in various forms (WPAR [8], pods [4], jails [9], etc.) but have recently become popularized through tools such as Docker [10] and LXC [11]. In their basic form, containers are simply a set of one or more processes that have been isolated from the rest of the operating system through some software mechanisms (generally in the kernel). Due to this logical grouping, containers are a convenient method of packaging software components, and can be used for scaling cloud applications by instantiating multiple instances of components (also known as microservices). Pre-copy causes serious problems for container migration that weren't evident with VMs. For containers, tracking memory involves quite a few interactions between the kernel and migration manager since there is no hypervisor to depend on [12]. Absolute migration time is also an issue [13] (measured from the time the migration command is issued, not just the downtime). In the case where program execution phase is the purpose of the migration, it is important to migrate as early as possible so as to maximize the benefit of the detected phase.

f) Page Fault Latency: The largest cost of post-copy migration is copying memory from the remote machine. A common method to copy memory during post-copy migration is by on-demand paging, which is latency sensitive. Commodity networking technology is not equipped to deal with this kind of scenario, as networks are optimized for high throughput bulk transfers over long links. In commonly deployed networks, latency considerations are secondary to guaranteed delivery and flexible routing options. More specialized solutions such as RDMA are known to be an order of magnitude faster than the ubiquitous TCP/IP over Ethernet [14], [15], but still 23x slower than local memory accesses[14]. The common handling of a page fault means loading content from an I/O device into memory, which is what the operating system is designed for. The slowest component on the datapath is the I/O device, with reads from spinning disks in the 5-10ms range, and flash-based devices being a order of magnitude faster. Reading memory remotely over a commodity network is considerably faster (30-100μs). Systems that use remote memory access over a network such as RamCloud [16] estimate that latency may reach 5-10μs (3 orders of magnitude faster than a spinning disk). However with new interconnect hardware that is becoming available on the market, access latency is likely to be less than 1μs [17], at which point software overhead imposed by the operating system becomes a serious issue.

g) Required Modifications: We are targeting a system composed of a rack of independent machines, all running the same operating system. The operating system is not modified to be single system image (like Popcorn Linux [18]), nor does it have any direct communication between kernels (like Barrelfish [19]). The only changes required in the operating system are in the way that it handles page faults, and a small driver for the new interconnect. That means minimal changes to commodity off-the-shelf operating systems, and existing data center installations. The changes to the operating system are centered around taking advantage of new hardware to supplement the existing page fault handling mechanism.

II. Concept

a) Hardware/Software Co-Design: We realize the importance of employing hardware/software co-design to ensure maximum cooperation between the components. The most effective way to ensure the lowest possible latency is to ensure all the components of the system are working together in harmony. Therefore we enumerate the requirements of the host CPU and operating system to influence the design of the interface exposed by the interconnect hardware. At the same time, we must also take into consideration how to efficiently handle the responses and error conditions in software that are generated by the interconnect.

A. Interconnect Characteristics

To understand the implications on systems software, we must first describe the required characteristics of the interconnect. The purpose of the interconnect is to transfer one page of memory from a remote machine to the local machine as quickly as possible, in response to a page fault. Note that during the time the system is resolving the page fault, the process is not doing any work, which causes lags in processing or external user requests to the application. The goal is to resolve the page fault as quickly as possible so as to minimize any impact it may have on application performance. In order to do so, our proposed interconnect would ideally connect to the platform's system bus directly, very much the same as the *Remote Memory Controller* described in soNUMA [20]. We expect that if this concept is proven useful, the controller would be integrated directly into the chip in the same way DDR or PCIe controllers are integrated today, precluding the need for an external NIC.

a) Rack Scale: We expect the majority of container migrations to take place at the rack-scale. Racks in a data center are often viewed as a logical unit, share a top-of-rack switch for network communication, and it has been shown that most traffic inside a cloud data center is local to the rack [21]. That means we must have an interconnect that can operate well with a node count of between 10 to 40. There are at least two viable options as shown in the SCI (Scalable Coherent Interface) spec [22], and any option that provides low latency connectivity may be used.

b) Communication between nodes: When the host copies a page of memory, we want to be sure that it arrives intact. Therefore, we must assume that the communication protocol employs error detection (such as CRC) and guaranteed delivery in the link layer. Since the communication is local, the protocol should optimize for the best case (assumes delivery) and any recovery (i.e. retransmits) can take a slower path.

c) Packet Format: Since there is no routing involved, the packet format can be very simple, saving the overhead of encoding and decoding many fields. For each process, We are establishing a globally distributed virtual address space, which means the page contents may exist on any machine in the cluster, and the virtual address of the page can be used to locate the data through a broadcast mechanism.

d) Virtual Addressing: Before each access by the interface card to memory, the virtual address is translated by an embedded MMU that enables the card to use an address space of a particular process. Thus, the card effectively works in the address space of the target application, and benefits from the existing page tables that are maintained by the operating system. This 'built-in MMU' makes setting up a shared region trivial, as the entire virtual address space of the application is automatically shared, once a process registers for migration. Since the ASID (address space identifier) is added to each request, the card can service page requests from multiple address spaces simultaneously.

Using virtual addresses also allows us to relax the requirement for pinning memory before DMA accesses. Since each virtual address is translated before access, it is possible that a mapping does not exist (page not present) meaning the virtual memory has been swapped out to disk, or (more likely) not yet been mapped to a physical page by the operating system. However, this relaxation requires hardware support on the card to be able to notify the OS of the problem, and wait for resolution. When a translation fails, the kernel is notified by the card for resolution by way of an interrupt. The local host (to which the card is attached) is responsible for handling this fault by having the operating system map a page in exactly the same manner as if the fault were caused by the CPU itself. Only one additional step is required, which is to have the kernel notify the card that the page fault has been resolved, and should retry the memory access. In our implementation, the local interface card will access the target address on purpose before the operating system completes the mapping. This will preload the mappings in the on-card cache, and put the card into a wait state for the kernel to complete the mapping. As soon as the mapping is complete, the kernel signals the card by writing a register, and the memory page is copied.

B. Host Characteristics

a) Page Ownership: If the CPU were to only *access* memory from a remote machine rather than transfer the page to the local machine (as is done in disaggregated memory architectures), the operating system would not need to be involved in the access, so a page fault would not need to be generated. However, all accesses would incur the penalty of the interconnect's latency, greatly increasing the overhead of memory accesses, slowing application execution[23]. Transferring ownership of the page to the local machine avoids communication overhead at the cost of a one-time page fault. Since the operating system is responsible for maintaining the list of free physical memory pages as well as virtual to physical page mappings for each address space, it must be involved in any changes requiring allocation or mapping of memory. That means for each page that we wish to transfer, the operating system must be responsible for the page fault resolution, because a new physical page must be allocated to hold the incoming data, and mapped to the correct virtual address. Handling the fault causes the processor to change modes, which can cost hundreds of cycles but that is an unavoidable cost that is necessary to pay in order to involve the operating system. We can reduce the cost by allocating pages to a page pool before a page fault occurs. Then at the time of the page fault, the kernel can take a page from the pool and map it to the faulting virtual address.

System requirements for process migration are simpler than general-purpose DSM (distributed shared memory) because *write* transactions do not need to be supported [24]. Supporting write transactions requires a much more complex protocol to ensure coherency across all participating nodes in the cluster, since multiple processors on multiple machines may write to the same memory page at the same time. With migration, a process can only be in a running state on one physical machine at a time, and we do not support the execution of different threads of the same process distributed across multiple machines. The goal is to migrate the necessary data to be close to the processors executing the process. We can then optimize for *read* transactions which the FaRM system showed to be 2x faster than writes[14]. That means all subsequent accesses to that page (even from multiple threads) are to local memory, at the full speed of the local memory bus.

b) Low latency: Hardware for DSM addresses latency by employing special-purpose I/O hardware (such as Dolphin PXH830) to give direct access to various sized memory windows of other physical machines. We believe future hardware will reduce this latency by at least one order of magnitude by using special purpose hardware designed for memory accesses rather than I/O. To take advantage of the low latency, we must modify the operating system to also support low latency operations. Page fault resolution in existing operating systems can take thousands of machine cycles to complete, depending on TLB misses, cache misses, etc. For a machine running at 4 GHz, 4000 cycles is approximately 1μs, which is longer than the time expected for the interconnect to retrieve the remote page [17]. Therefore it is imperative to minimize the operations performed by the operating system during remote page resolution, and to hide the latency by performing actions in parallel as much as possible.

c) Implicit communication: One of the drawbacks of existing memory sharing models is that they are aimed at solving the problem of distributed shared memory at the application level, which requires the explicit involvement of the application. For process migration, this is problematic for three reasons: first we must write the application with explicit knowledge of the communication; second, the further involvement of software (calling a I/O function) hurts the latency as compared to intrinsic memory load/store operations; and third, lack of system support means each application must be set up separately to use this mechanism. What we want is to have all support for moving memory in the infrastructure, so any application can be supported transparently without modifying the application, or burdening the application programmer.

d) Cache Coherency: Cache coherency in a cc-NUMA system ensures that all processors have a consistent view of shared memory. In the case of process migration, cache coherency can be used if a process is to be migrated back to its original machine. It is an optimization that can further reduce latency by invalidating pages on the source machine that have been written on the target machine. After the migration back, the pages that were transferred and modified must be transferred again. Other pages that were unused or only read, remain valid. Since physical pages are already mapped to these virtual addresses, we note that the operating system no longer needs to be involved in the transfer, which can be automatic.

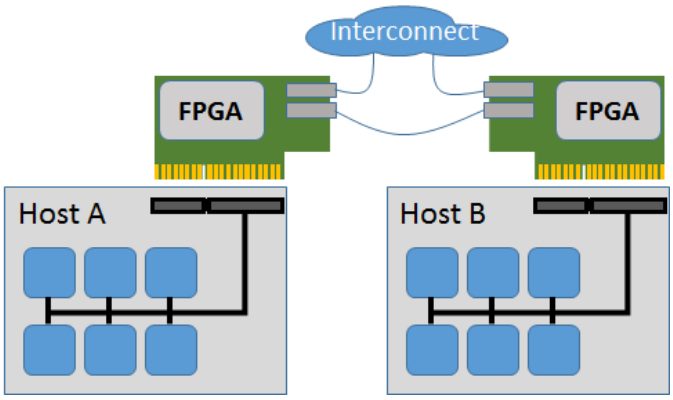

Fig. 2. Architecture diagram showing the cache coherent bus between CPU cores and FPGA, and the prototype scalable interconnect between FPGAs.

III. Design

The interconnect is designed to be an extension of the system bus, for exclusive use by the operating system for the purposes of container migration. Since the interconnect must be involved during the handling of page faults by the operating system, all code to manage the interconnect resides in the kernel (at least, with a monolithic kernel design such as Linux). We first describe a prototype interconnect that is currently being designed to prove the concept and further develop the page fault mechanism. Finally, we describe the details required for setting up a process for migration, followed by the steps taken when resolving a page fault.

A. Interconnect Prototype

A prototype of the NIC [25] is being built for evaluation as part of the OPERA EU project. For simplicity, the initial prototype involves only two machines connected with FPGA equipped CAPI expansion cards (Nallatech 385A-SoC [26]). The FPGA is used to implement the high-speed interconnect logic. The local interface card is connected to the remote interface card over a point-to-point 10Gb/s link using the SerialLite protocol over fiber optics. The card is equipped with two such ports, which allows for expansion into a ring topology in the future (as shown in figure 2). All higher level communication (network layer) is through a proprietary protocol consisting of asynchronous requests and responses. Memory read requests are initiated by the local interface card (target) by sending a memory read message to the remote host.

a) Addressing: To handle a remote page fault, there must be a common addressing scheme in place to be able to identify the correct memory page that should be copied. The two key pieces of information are the memory address and the address space. The memory address is simply the virtual address that caused the page fault. Since we are essentially dealing with a single process that is distributed among multiple servers for a short time, the virtual address can remain constant across all servers. For a concrete example, if we know that a migrated process on the target server generates a page fault at address 0x10001000, then we must request this exact same address on the source server. This realization is the key to being able to offload nearly all support into hardware, and make the system fast. So once the interface cards use virtual addresses, it is a relatively simple matter for us to be able to transfer a particular virtual address, without modification, from the target server to the source server to transfer the memory. The second piece of information is a bit more challenging. The address space refers to the virtual memory subsystem in which each process has its own continuous range of memory addresses, but is sparsely populated (i.e. not every address is mapped to a physical memory location). In the operating system, each address space is associated with a particular process and is referenced by an address space identifier (ASID). The ASID is primarily used by the processor for setting up and maintaining page tables and caches. The assignment of the ASID to a process is under the control of the operating system. In fact, two local processes may share the same ASID (in which case they share the same virtual memory) and are better known as threads. The assignments are unique to a particular server, and no server should know or should need to know how ASIDs relate to processes on any other server. Even if we were to decide that this knowledge could improve container migration, it is not practical for each server to know about all of the ASIDs of all of the other servers. The number of ASIDs would grow exponentially with the number of servers, and just the communication to manage them all would be overwhelming even for a small number of servers. Luckily, we can reduce the problem to include only ASIDs related to processes that are currently migrating (i.e. all of the processes of a particular container), and the number of servers that need to know these ASIDs to only the set of servers involved in the migration (which may be more than two as we will see shortly). This leaves us with a relatively short list that needs to be updated only when a migration begins or ends. This list can be offloaded to the interface card as well, further simplifying the software, and minimizing the latency of a given page fault.

b) Interface Card: Unfortunately, externally accessible hardware that is directly attached to the system bus does not exist in commodity servers. In order to build a working prototype for testing with real hardware available today, we are looking at the best option available, which is the CAPI protocol as implemented in the IBM POWER8® and OpenPOWER systems. CAPI is intended for providing a cache-coherent view of system memory with attached accelerators over PCIe (I/O bus). That means we can get the required functionality of accessing memory through virtual addresses, but the physical layer is less than ideal in terms of latency since it relies on PCIe rather than being directly attached to the system bus. The design of the FPGA on the expansion card (accelerator) can be described as a combination of several hardware blocks in two parts - The POWER Service Layer (PSL) and the Accelerator Function Unit (AFU). The AFU is used to implement specific functionality behind the PSL. The PSL, among other things, provides memory address translation services to allow each AFU direct access to user space memory.

B. Page Fault Mechanism

a) Setup: Before a process can be migrated, it must be registered so that the interconnect is notified of the address space used by the process. We will call this the *local address space identifer* (local ASID). The local ASID is unique to that particular host and is used by the local interface card when accessing memory on the host by virtual address. When the process is registered, the local interface card must translate

this local ASID to a globally unique ASID, and hold it in a lookup table on the card. Any communication by the interface cards over the interconnect must use the global ASID. Before an interface card can initiate communication with a local host, it must first look up the local ASID to know how to find the correct page table for virtual-to-physical address translation.

b) Remote Page Fault Resolution: At a high level, the page fault resolution is composed of two main actions: retrieval of the remote page, and mapping the contents into the address space of the faulting process. The mapping may take some time, in fact it is likely to be longer than the round trip on the interconnect to copy the missing page. To help hide the latency, the two actions should be performed concurrently. Fig. 3 shows the sequence of steps taken to copy a remote memory page. (1) When a migrated process attempts to access a memory location that has not yet been copied (but is otherwise valid), the access generates a page fault. (2) The fault causes the processor to switch to a privileged mode of execution which gives the operating system an opportunity to handle the fault. (5) For remote page faults, the kernel must first mark the thread as blocked (pending page fault resolution), allocate a new physical memory page (or draw from the page pool) and map it to the virtual address that caused the fault. The physical page must be mapped before the local interface card can copy remote data into the host's memory. (3) After the local interface card receives notification of the missing page, it sends a memory read request to the remote interface card, specifying the virtual address, global ASID, and number of pages to read. (4) The remote interface card looks up the corresponding local ASID, and performs a DMA read on the requested virtual address from the host's memory. (6) When the DMA read completes, the remote interface card fulfills the request by returning the data with a sequence number to the local interface card. (8) The local interface card performs a DMA write to put the data in the memory of the local host, and then notifies the local kernel of completion. (7) By this time, the kernel should have finished mapping the physical page into the destination address space. If the DMA write is attempted before the kernel completes the mapping, the MMU of the local interface card will not be able to translate the virtual address for the DMA request, and instead will notify the kernel and stall. The kernel can notify the local interface card upon completion of the mapping, and the DMA transfer will be retried. (9) When the local kernel receives notification that the DMA write is complete, it can complete the page fault resolution by setting the blocked thread as ready-to-run.

IV. Related Work

a) Hardware Developments: Recently, the GenZ[27] specification has been drafted, which is intended to be a architecture-agnostic, high-speed, low-latency interconnect for attaching memory-mapped devices to the system bus. Such a bus would make an excellent candidate for building a controller for handling remote page faults between machines. The GenZ consortium is composed of leading industrial companies, which is a sign that there is a need for such an interconnect in commercial systems. Similarly, specs for CCIX[28] and OpenCAPI [29] have been drafted and announced. These both specify ISA-agnostic cache coherency protocols for accelerators. We can envision building an interconnect interface card that connects to the host using GenZ, and keeps cache

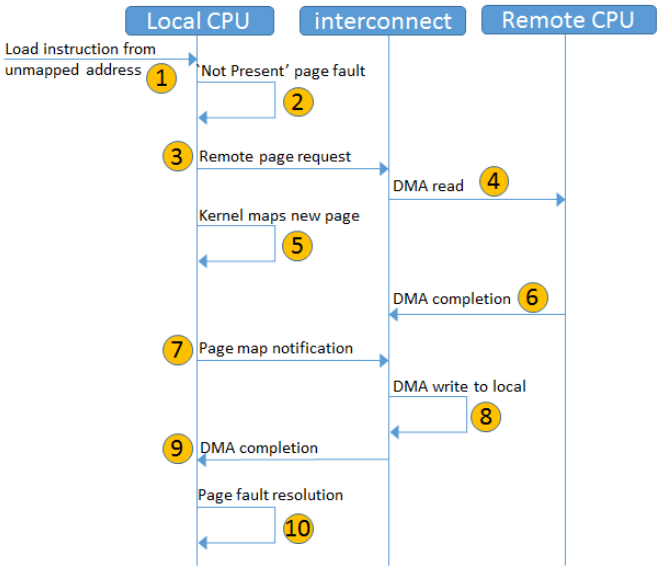

Fig. 3. Sequence diagram showing the high level steps when handling a page fault.

coherency between its own host and other hosts through CCIX or OpenCAPI to implement remote page reads. This may take the form of a remote memory controller (RMC) as described in soNUMA [20].

b) Distributed Memory: All DSM systems to date aim to solve the problem of remotely accessing memory for the purpose of multi-processing – that is, having multiple distributed threads share an address space across physical machines communicating over some interconnect. Disaggregated memory [23] places RAM in a remote machine, which also requires read and write accesses. While our goals are different, the hardware developments may be mutually beneficial.

c) Migration: We have a lot in common with published work on process migration in the past 30 years [6], [30], [5]. We keep 3 out of 4 requirements specified by Zap [4], only breaking the 3rd which specifies the source machine should not continue to serve the target machine. This is an artifact of post-copy migration, which can have a bounded time (i.e. we can force movement of all memory pages if needed).

V. Conclusion

Post-copy context migration is a powerful tool that data center operators can use to balance server load at run-time. The main issue with post-copy migration is the latency incurred when serving remote page faults. Architectural developments may soon allow us direct access to the system bus, allowing communication between machines at speeds approaching that of the memory bus. The ability of the operating system to be able to handle page faults with very low latency is becoming important in order to take full advantage of new interconnects so that we can build scalable, composable systems.

Acknowledgements

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 688386.

REFERENCES

[1] A. Barak, A. Shiloh, and L. Amar, "An organizational grid of federated mosix clusters," in *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid - Volume 01*, ser. CCGRID '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 350–357. [Online]. Available: http://dl.acm.org/citation.cfm?id=1169222.1169488

[2] C. Isci, J. Liu, B. Abali, J. O. Kephart, and J. Kouloheris, "Improving server utilization using fast virtual machine migration," *IBM J. Res. Dev.*, vol. 55, no. 6, pp. 365–376, Nov. 2011. [Online]. Available: http://dx.doi.org/10.1147/JRD.2011.2167775

[3] A. Venkat and D. M. Tullsen, "Harnessing isa diversity: Design of a heterogeneous-isa chip multiprocessor," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 121–132. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665692

[4] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The design and implementation of zap: A system for migrating computing environments," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 361–376, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844162

[5] A. Reber, "Process migration in a parallel environment," Ph.D. dissertation, 2016. [Online]. Available: http://dx.doi.org/10.18419/opus-8791

[6] E. Zayas, "Attacking the process migration bottleneck," *SIGOPS Oper. Syst. Rev.*, vol. 21, no. 5, pp. 13–24, Nov. 1987. [Online]. Available: http://doi.acm.org/10.1145/37499.37503

[7] "Vmware vsphere vmotion architecture, performance and best practices in vmware vsphere," 2011. [Online]. Available: http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-vsphere51-vmotion-performance-white-paper.pdf

[8] *Workload Partitioning (WPAR) in AIX 6.1*, 2008. [Online]. Available: https://www.ibm.com/developerworks/aix/library/au-wpar61aix/

[9] P. H. Kamp and R. N. M. Watson, "Jails: Confining the omnipotent root," in *In Proceedings of the 2nd International SANE Conference*, 2000. [Online]. Available: http://phk.freebsd.dk/pubs/sane2000-jail.pdf

[10] *Docker - Build, Ship, Run*. [Online]. Available: https://www.docker.com/

[11] *LinuxContainers.org Infrastructure for container projects*. [Online]. Available: https://linuxcontainers.org/

[12] P. Emelyanov, "Live migrating a container: Pros, cons and gotchas," 2015. [Online]. Available: https://www.slideshare.net/Docker/live-migrating-a-container-pros-cons-and-gotchas

[13] "Combining pre-copy and post-copy migration," 2016. [Online]. Available: https://lisas.de/~adrian/?p=1253

[14] 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)*. Seattle, WA: USENIX Association, 2014, pp. 401–414. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/dragojevi{\'c}

[15] C. Mitchell, Y. Geng, and J. Li, "Using one-sided rdma reads to build a fast, cpu-efficient key-value store," in *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*. San Jose, CA: USENIX, 2013, pp. 103–114. [Online]. Available: https://www.usenix.org/conference/atc13/technical-sessions/presentation/mitchell

[16] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, "The case for ramclouds: Scalable high-performance storage entirely in dram," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1713254.1713276

[17] *Dolphin PXH830 Host Adapter*, 2015. [Online]. Available: http://www.dolphinics.com/products/PXH830.html

[18] A. Barbalace, M. Sadini, S. Ansary, C. Jelesnianski, A. Ravichandran, C. Kendir, A. Murray, and B. Ravindran, "Popcorn: Bridging the programmability gap in heterogeneous-isa platforms," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15. New York, NY, USA: ACM, 2015, pp. 29:1–29:16. [Online]. Available: http://doi.acm.org/10.1145/2741948.2741962

[19] A. Baumann, S. Peter, A. Schüpbach, A. Singhania, T. Roscoe, P. Barham, and R. Isaacs, "Your computer is already a distributed system. why isn't your os?" in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, ser. HotOS'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 12–12. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855568.1855580

[20] S. Novakovic, A. Daglis, E. Bugnion, B. Falsafi, and B. Grot, "Scale-out numa," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 3–18, Feb. 2014. [Online]. Available: http://doi.acm.org/10.1145/2654822.2541965

[21] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879175

[22] *1596-1992 - IEEE Standard for Scalable Coherent Interface (SCI)*. IEEE, 1992. [Online]. Available: http://ieeexplore.ieee.org/document/347683/references

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

[24] R. F. Lyerly, "Popcorn linux: A compiler and runtime for state transformation between heterogeneous-ISA architectures," Ph.D. dissertation, 2016. [Online]. Available: http://www.ssrg.ece.vt.edu/theses/PhdProposal_Lyerly.pdf

[25] J. Nider, Y. Binyamini, and M. Rapoport, "Remote page faults with a capi based fpga," in *Proceedings of the 10th ACM International Systems and Storage Conference*, ser. SYSTOR '17. New York, NY, USA: ACM, 2017, pp. 20:1–20:1. [Online]. Available: http://doi.acm.org/10.1145/3078468.3078489

[26] *Nallatech 385A-SoC System on Chip FPGA Accelerator Card*. [Online]. Available: http://www.nallatech.com/store/fpga-accelerated-computing/pcie-accelerator-cards/nallatech-385a-soc/

[27] "The GenZ consortium," 2016. [Online]. Available: http://genzconsortium.org/about/

[28] "Cache coherent interconnect for accelerators (ccix)," 2016. [Online]. Available: http://www.ccixconsortium.com/

[29] "Opencapi consortium," 2016. [Online]. Available: http://opencapi.org/

[30] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch, "The sprite network operating system," *Computer*, vol. 21, no. 2, pp. 23–36, Feb. 1988. [Online]. Available: http://dx.doi.org/10.1109/2.16