Revisiting RDMA Buffer Registration in the Context of Lightweight Multi-kernels

Balazs Gerofi, Masamichi Takagi and Yutaka Ishikawa
RIKEN Advanced Institute For Computational Science, JAPAN
{bgerofi, masamichi.takagi, yutaka.ishikawa}@riken.jp

ABSTRACT

Lightweight multi-kernel architectures, where HPC specialized lightweight kernels (LWKs) run side-by-side with Linux on compute nodes, have received a great deal of attention recently due to their potential for addressing many of the challenges system software faces as we move towards exascale and beyond. LWKs in multi-kernels implement only a limited set of kernel functionality and the rest is supported by Linux, for example, device drivers for high-performance interconnects. While most of the operations of modern high-performance interconnects are driven entirely by user-space, memory registration for remote direct memory access (RDMA) usually involves interaction with the Linux device driver and thus comes at the price of service offloading.

In this paper we introduce various optimizations for multi-kernel LWKs to eliminate the memory registration cost. In particular, we propose a safe RDMA pre-registration mechanism combined with lazy memory unmapping in the LWK. We demonstrate up to two orders of magnitude improvement in RDMA registration latency and up to 15% improvement on `MPI_Allreduce()` for large message sizes.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: Organization and Design

Keywords

Operating Systems; Multi Kernels; RDMA Registration

1. INTRODUCTION

With the increasing complexity of high-end supercomputers, there is a growing consensus in the system software community that the current software stack will face significant challenges as we look forward to exascale and beyond. The necessity to deal with extreme degree of parallelism, heterogeneous architectures, multiple levels of memory hierarchy, power constraints, etc. advocates operating systems that can rapidly adapt to new hardware requirements, and that can support novel programming paradigms and runtime systems. On the other hand, a new class of more dynamic and complex applications are also on the horizon, with an increasing demand for application constructs such as in-situ analysis, workflows, elaborate monitoring and performance tools [20, 16]. This complexity relies not only on rich features of POSIX, but also on the Linux APIs (such as the `/proc`, `/sys` filesystems, etc.) in particular.

While the traditional "standalone" lightweight kernels have proven successful in tackling the high degree of parallelism so that scalable performance for bulk synchronous applications can be delivered, they generally fail to provide a fully Linux compatible environment [11, 13, 19, 8, 7, 2]. An alternative hybrid approach recognized recently by the system software community is to run Linux simultaneously with a lightweight kernel on compute nodes and multiple research projects are now pursuing this direction [17, 1, 14, 3]. The basic idea is that simulations run on an HPC tailored lightweight kernel, ensuring the necessary isolation for noiseless execution of parallel applications, but Linux is leveraged so that the full POSIX API is supported. Additionally, the small code base of the LWK can also facilitate rapid prototyping for new, exotic hardware features [4, 5].

Linux compatibility is often provided by offloading OS requests (e.g., system calls) from the LWK to Linux. Some of the typically offloaded operations are file I/O, Linux specific APIs (i.e., the `/proc`, `/sys` filesystems), and access to Linux device drivers, such as drivers for high-performance interconnects. Although modern interconnects, such as Infiniband, enable applications to drive the NIC directly from user-space using regular `load`/`store` instructions, certain functionality still requires interaction with the device driver. One particular example is registration of memory buffers for remote direct memory access (RDMA) operations. Because the Infiniband driver is invoked for RDMA registration via a `write()` call, in multi-kernel settings an additional cost for offloading is also involved.

In this paper, we investigate how to minimize the cost of RDMA registration in the context of lightweight multi-kernels and implement our proposal in the IHK/McKernel architecture [14, 4]. In summary, we make the following contributions:

- We propose a safe RDMA pre-registration mechanism that registers McKernel's physical memory during the proxy process initialization in Linux, without exposing the RDMA memory mappings to the LWK process itself;
- We make MVAPICH2 RDMA pre-registration aware

EuroMPI'16, September 25-28, 2016, Edinburgh, United Kingdom
 ISBN 978-1-4503-4234-6/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2966884.2966888

so that registration requests can be served locally, avoiding the otherwise expensive `write()` calls.

- Finally, we further eliminate offloading cost by introducing a lazy `munmap()` synchronization mechanism, which postpones reflecting unmap operations on the Linux side until it is absolutely necessary and performs them by piggybacking unmap requests to consecutive offloaded system calls.

Through microbenchmarks, we demonstrate up to two orders of magnitude improvement in RDMA registration latency and up to 15% improvement on `MPI_Allreduce()` for large buffers.

The rest of this paper is organized as follows. We begin with providing some background information on hybrid kernels and IHK/McKernel in Section 2. Section 3 discusses RDMA pre-registration and required MPI modifications. Experimental evaluation is given in Section 4. Section 5 surveys related work, and finally, Section 6 presents future plans and concludes the paper.

2. BACKGROUND

We first provide an overview of IHK/McKernel [14, 4], our hybrid kernel configuration upon which the rest of this work is built.

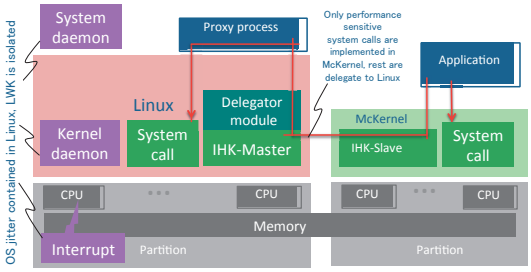


Figure 1: Overview of the IHK/McKernel architecture and the system call delegation mechanism.

As we described earlier, lightweight multikernels are relatively new in HPC system software and multiple recent research projects are now investigating this direction [14], [1], [17], [3]. IHK/McKernel is our solution to providing such a hybrid configuration. The IHK/McKernel architecture is shown in Figure 1. At the heart of the stack is a low-level software infrastructure called Interface for Heterogeneous Kernels (IHK) [14]. IHK is a general framework that provides capabilities for partitioning resources in a many-core environment (e.g., CPU cores and physical memory) and it enables management of lightweight kernels. It also provides an Inter-Kernel Communication (IKC) layer, upon which system call delegation is implemented. McKernel is a lightweight kernel designed for HPC, which can be booted from IHK.

In case of IHK/McKernel, the application is primarily run on McKernel to achieve the desired scalability and reliability, but McKernel implements only performance sensitive system calls and the rest of the OS services are offloaded to Linux. With respect to this study, the most important attributes of McKernel are its system call offloading and address space management mechanisms.

3. DESIGN AND IMPLEMENTATION

Before diving into RDMA pre-registration, we begin with a more detailed description of the system call offloading mechanism, which is illustrated in Figure 1. During system call delegation McKernel marshalls the system call number along with its arguments and sends a message to Linux via a dedicated IKC channel. The corresponding proxy process running on Linux is by default waiting for system call requests through an `ioctl()` call into IHK's system call delegator kernel module. The delegator kernel module's IKC interrupt handler wakes up the proxy process, which returns to userspace and simply invokes the requested system call. Once it obtains the return value, it instructs the delegator module to send the result back to McKernel, which subsequently passes the value to user-space.

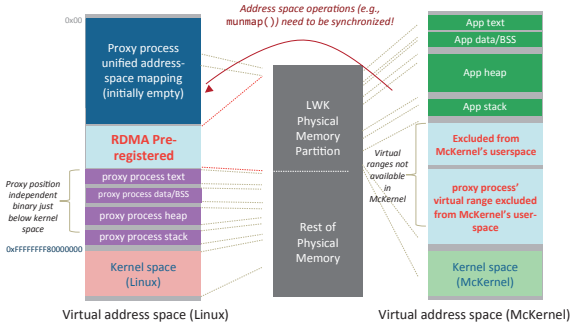


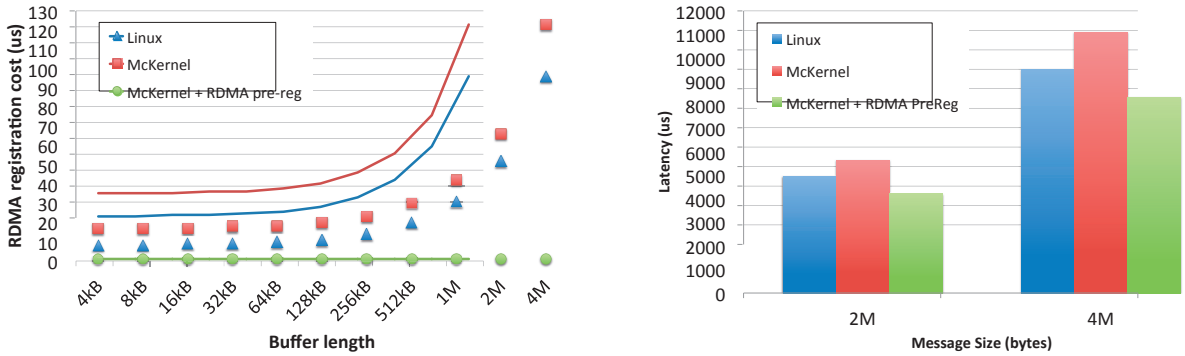
Figure 2: Unified address space between Linux and McKernel with RDMA pre-registration.

An addition to the system call offloading mechanism is the concept of unified address space model. IHK/McKernel ensures that offloaded system calls can seamlessly resolve arguments even in case of pointers without explicitly transferring data the pointers refer to. The unified address space model, along with the RDMA pre-registration (which we will detail below), is depicted in Figure 2 and it is implemented as follows. First, the proxy process is compiled as a position independent binary, which enables us to map the code and data segments specific to the proxy process to an address range which is explicitly excluded from McKernel's user space. Second, the entire valid virtual address range of McKernel's application user-space is covered by a special mapping in the proxy process for which we use a pseudo file mapping in Linux. This mapping is indicated by the blue box on the left side of the figure.

Note, that the proxy process does not need to fill in any virtual to physical mappings at the time of creating the pseudo mapping and it remains empty unless an address is referenced. Every time an unmapped address is accessed, the page fault handler of the pseudo mapping consults the page tables corresponding to the application on the LWK and maps it to the exact same physical page[1]. Needless to say, Linux' page table entries in the pseudo mapping have to be occasionally synchronized with McKernel, for instance, when the application calls `munmap()` or modifies certain mappings.

An overview of RDMA pre-registration is also depicted by Figure 2. The basic idea is to map the LWK dedicated phys-

[1]For further details on unified address space refer to [6]



(a) RDMA registration cost.

(b) `MPI_Allreduce()` using large buffers.

Figure 3: RDMA registration cost and `MPI_Allreduce()` using large buffers on 16 compute nodes.

ical memory into the proxy process at the time of launching the application and to register the entire region for RDMA capability with the NIC. This mapping is indicated by the grey box on the left side of the figure. As it's also shown, the corresponding virtual range is excluded from the actual LWK process, which in turn ensures that regular application code can not access this region of the memory directly. The registration details are transferred to McKernel at LWK process creation time and are utilized as follows.

When the application on the LWK requests to register a memory region McKernel can resolve the buffer's virtual address into physical space and calculate its offset in the LWK dedicated physical partition. Instead of registering a new RDMA region, it can simply translate the request into the pre-registered area and return the associated key with the computed offset. Note that McKernel ensures that `MAP_ANONYMOUS` memory mappings are contiguous in physical memory, otherwise the offset calculation would not be viable. At the OS API level we have introduced a new system call which expects a pair of virtual address plus buffer length and returns the RDMA registration keys as well as the corresponding offset.

We will now turn our attention towards another system call offload optimization which we devised in conjunction with RDMA pre-registration. As we mentioned earlier, certain address space operations need to be reflected in the proxy process' address space to ensure consistency. One particular example is the `munmap()` system call. However, since the proxy process only executes offloaded system calls, it is sufficient to update its view of the virtual address space just at the time when it executes a call that could possibly refer to user-space addresses. Therefore, to eliminate the communication cost associated with unmap operation we simply record the memory range and piggyback the operation to the subsequent offloaded system call.

3.1 MPI Integration

This section discusses the necessary modifications to MVAPICH2 so that RDMA pre-registration can be exploited. There have been mainly two set of modifications made. First, we modified the `register_memory()` and `deregister_memory()` functions so that instead calling the IB library MVAPICH invokes the specific McKernel system call. In order to stay consistent with the rest of the MPI library, the call doesn't change the virtual address of the registration, however, it records the pre-registered RDMA keys. Only at the time of issuing the actual RDMA operation is the virtual address rewritten to the one corresponding to the pre-registered offset (e.g., in the `mv2_shm_coll_prepare_post_send()` function).

A similar set of modifications have been made to the functions dealing with `vbuf` structures. Some of the most important ones are `VBUF_SET_RDMA_ADDR_KEY()` and `vbuf_init_rdma_write()`.

4. EVALUATION

This section provides evaluation of the proposed mechanism using various micro-benchmarks.

4.1 Experimental Setup

Our experiments were conducted on a small cluster where each node consists of Intel Xeon Ivy Bridge (E5-2670 v2 @ 2.50GHz) CPUs with two sockets, ten cores per socket and two hardware threads per core. The nodes are equipped with 64GB RAM and with Mellanox Infiniband MT27500 (ConnectX-3) interconnection network.

4.2 Results

The first experiment we conducted is to measure RDMA registration cost as the function of buffer size. To assess both the original offload cost and the benefits of the proposed mechanism we measure three configurations, Linux with un-faulted virtual ranges, McKernel with regular offload, and McKernel with pre-registration. Figure 3a indicates the results. As seen, McKernel's offloaded registration mechanism adds a constant overhead compared to the baseline Linux value. On the other hand, pre-registration on McKernel yields up to two orders of magnitude better performance than the baseline when applied to large buffers.

The second experiment is `MPI_Allreduce()` on 16 compute nodes using MVAPICH2 with its default configuration parameters. We observed that `MPI_Allreduce()` allocates internal buffers in each iteration of the benchmark which in turn get registered for RDMA. By applying the proposed pre-registration optimization these requests can be eliminated and consequently the overall performance is improved. We report values for two large buffer sizes where this effect proved significant. As seen in Figure 3b, we obtain 15% and 12% improvements on allreduce operations using 2MB and 4MB message sizes, respectively.

5. RELATED WORK

Various issues related to RDMA registration have been studied previously. Mietke et al. provided a detailed breakdown of RDMA registration cost on Infiniband networks [10]. Woodall et al. described a pipeline protocol to overlap memory registration with RDMA operations [18]. Shipman et al. investigated network buffer utilization and introduced a new protocol to increase the efficiency of receiver buffer utilization for Infiniband [15]. Dong et al. proposed a helper thread based memory registration/deregistration strategy to reduce registered memory on multicore architectures [9]. Ou et al. introduce various optimizations to reduce the overhead of communication buffer management [12].

While these works also focus on the problem of how to reduce cost of RDMA registration, none of them considers multi-kernel settings. To the best of our knowledge this is the first proposal which explicitly aims at eliminating offload cost in lightweight multi-kernel architectures.

6. CONCLUSION AND FUTURE WORK

In this paper we have introduced various optimizations for multi-kernel LWKs to eliminate the system call offloading cost associated with RDMA memory registration. Specifically, we have proposed a safe RDMA pre-registration mechanism and combined it with lazy memory unmapping. We have implemented the proposed mechanism in the McKernel lightweight multi-kernel and integrated support into MVAPICH2. We demonstrated up to two orders of magnitude improvement in RDMA registration latency as well as up to 15% improvement on `MPI_Allreduce()` when exchanging large messages.

In the future, we will further investigate techniques to reduce system call offloading cost in multi-kernel architectures.

Acknowledgment

This work is partially funded by MEXT's program for the Development and Improvement for the Next Generation Ultra High-Speed Computer System, under its Subsidies for Operating the Specific Advanced Large Research Facilities.

We also acknowledge the McKernel development efforts of Tomoki Shirasawa and Gou Nakamura from Hitachi.

7. REFERENCES

[1] Argo: An Exascale Operating System (Accessed: Jan, 2015). http://www.mcs.anl.gov/project/argo-exascale-operating-system.

[2] Kitten: A Lightweight Operating System for Ultrascale Supercomputers (Accessed: Sep, 2015). https://software.sandia.gov/trac/kitten.

[3] Brightwell, R., Oldfield, R., Maccabe, A. B., and Bernholdt, D. E. Hobbes: Composition and Virtualization As the Foundations of an Extreme-scale OS/R. In *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers* (New York, NY, USA, 2013), ROSS '13, ACM, pp. 2:1–2:8.

[4] Gerofi, B., Shimada, A., Hori, A., and Ishikawa, Y. Partially Separated Page Tables for Efficient Operating System Assisted Hierarchical Memory Management on Heterogeneous Architectures. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on* (May 2013), pp. 360–368.

[5] Gerofi, B., Shimada, A., Hori, A., Masamichi, T., and Ishikawa, Y. CMCP: A Novel Page Replacement Policy for System Level Hierarchical Memory Management on Many-cores. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing* (New York, NY, USA, 2014), HPDC '14, ACM, pp. 73–84.

[6] Gerofi, B., Takagi, M., Nakamura, G., Shirasawa, T., Hori, A., and Ishikawa, Y. On the Scalability, Performance Isolation and Device Driver Transparency of the IHK/McKernel Hybrid Lightweight Kernel. In *IEEE International Parallel and Distributed Processing Symposium* (2016), IPDPS '16.

[7] Giampapa, M., Gooding, T., Inglett, T., and Wisniewski, R. W. Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (Washington, DC, USA, 2010), SC '10, IEEE Computer Society, pp. 1–10.

[8] Kelly, S. M., and Brightwell, R. Software architecture of the light weight kernel, Catamount. In *In Cray User Group* (2005), pp. 16–19.

[9] Li, D., Cameron, K. W., Nikolopoulos, D. S., de Supinski, B. R., and Schulz, M. Scalable Memory Registration for High Performance Networks Using Helper Threads. In *Proceedings of the 8th ACM International Conference on Computing Frontiers* (New York, NY, USA, 2011), CF '11, ACM, pp. 38:1–38:10.

[10] Mietke, F., Rex, R., Baumgartl, R., Mehlan, T., Hoefler, T., and Rehm, W. Analysis of the memory registration process in the mellanox infiniband software stack. In *Euro-Par 2006 Parallel Processing*. Springer, 2006, pp. 124–133.

[11] Oral, S., Wang, F., Dillow, D. A., Miller, R., Shipman, G. M., Maxwell, D., Henseler, D., Becklehimer, J., and Larkin, J. Reducing Application Runtime Variability on Jaguar XT5. In *Proceedings of CUG'10* (2010).

[12] Ou, L., He, X., and Han, J. An Efficient Design for Fast Memory Registration in RDMA. *J. Netw. Comput. Appl. 32*, 3 (May 2009), 642–651.

[13] Pritchard, H., Roweth, D., Henseler, D., and Cassella, P. Leveraging the Cray Linux Environment Core Specialization Feature to Realize MPI Asynchronous Progress on Cray XE Systems. In *Proceedings of CUG'12* (2012).

[14] Shimosawa, T., Gerofi, B., Takagi, M., Nakamura, G., Shirasawa, T., Saeki, Y., Shimizu, M., Hori, A., and Ishikawa, Y. Interface for Heterogeneous Kernels: A Framework to Enable Hybrid OS Designs targeting High Performance Computing on Manycore Architectures. In *High Performance Computing (HiPC), 2014 21st Int. Conf. on High Performance Computing* (Dec 2014), HiPC '14, pp. 1–10.

[15] Shipman, G. M., Brightwell, R., Barrett, B., Squyres, J. M., and Bloch, G. Investigations on infiniband: Efficient network buffer utilization at scale. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2007, pp. 178–186.

[16] Tiwari, D., Boboila, S., Vazhkudai, S. S., Kim, Y., Ma, X., Desnoyers, P. J., and Solihin, Y. Active Flash: Towards Energy-efficient, In-situ Data Analytics on Extreme-scale Machines. In *Proceedings of FAST'13* (Berkeley, CA, USA, 2013), USENIX Association, pp. 119–132.

[17] Wisniewski, R. W., Inglett, T., Keppel, P., Murty, R., and Riesen, R. mOS: An Architecture for Extreme-scale Operating Systems. In *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers* (New York, NY, USA, 2014), ROSS '14, ACM, pp. 2:1–2:8.

[18] Woodall, T. S., Shipman, G. M., Bosilca, G., Graham, R. L., and Maccabe, A. B. High performance RDMA protocols in HPC. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2006, pp. 76–85.

[19] Yoshii, K., Iskra, K., Naik, H., Beckmanm, P., and Broekema, P. C. Characterizing the Performance of Big Memory on Blue Gene Linux. In *Proceedings of the 2009 International Conference on Parallel Processing Workshops* (2009), ICPPW '09, IEEE Computer Society, pp. 65–72.

[20] Zhang, F., Docan, C., Parashar, M., Klasky, S., Podhorszki, N., and Abbasi, H. Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform. In *Proceedings of IPDPS'12* (May 2012), pp. 1352–1363.