UMap: A user level memory mapping library

Marty McFadden Keita Iwabuchi Eric Green
Roger Pearce Maya Gokhale
Lawrence Livermore National Laboratory, Livermore, CA
{mcfadden8}@llnl.gov

Kai Wu Dong Li
UC Merced, Merced CA
{kwu42}@ucmerced.edu

I. Introduction

Memory mapping provides a convenient mechanism to associate files or regions in a backing store with a memory area. With the ubiquity of low latency, high bandwidth NVMe-attached SSDs, large data sets stored on these devices can be mapped into memory, simplifying out-of-core data analysis. Linux system mmap provides excellent performance in many use cases such as loading dynamic libraries, and over recent kernel releases can also work well with out of core access patterns.

The system level mmap service is tuned to perform reliably and consistently over a broad range of workloads, and thus lack customizability to characteristics of specific application access patterns. In prior work, we developed a data intensive memory map runtime as a loadable Linux kernel module. The runtime was optimized to data intensive out-of-core applications, and was used to search graph data sets as well as simulation data stored on node-local NVRAM [1], [2]. The module performed well on out-of-core data intensive applications, but required kernel modifications and frequent updates to retain compatibility with the fast-moving Linux kernel.

In this work, we have developed a user level library to support application-specific, customized memory mapping of arbitrary regions in a process's address space. As a user mode library, UMap avoids need to track kernel updates. To mitigate the cost of user mode page fault handling, UMap avoids the overhead of signal fault handlers by employing the userfaultfd [3] protocol released in Linux 4.3+ kernels. UMap is open source and available at https://github.com/LLNL/umap.

II. Userfaultfd

Userfaultfd is an asynchronous method to notify a user process of a page fault and to support atomic resolution of the page fault. The ioctl system call is used to register the address range and associated user space handler. Page faults in that range trigger messages from the kernel to the handler, who then can resolve the page fault in an application specific manner.

Userfaultfd was developed to support postcopy live migration of virtual machine pages from a memory server to a compute node. Postcopy migration allows a VM to start on the compute node before all its pages are transferred from the memory server. As the VM generates page faults for missing pages, the associated userfaultfd handler fetches requested pages over the network and used the atomic UFFDIO_COPY memcpy to put them into the VM address space [4]. Other potential applications include the FluidMem memory server [5], efficient snapshotting, CRIU lazy restore from disk, and distributed shared memory.

III. UMap handler overview

In contrast to such cloud computing scenarios, the UMap handler is designed for out-of-core data analysis of memory mapped data sets.

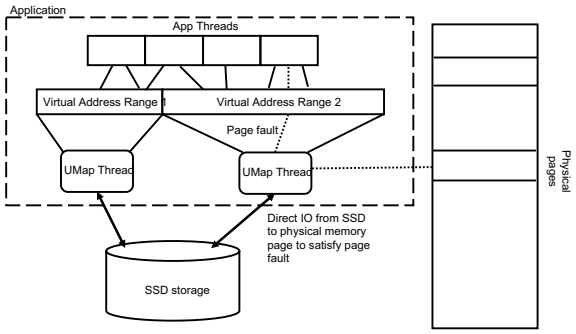

Fig. 1: UMap architecture

As shown in Figure 1, an application contains UMap handler threads and application threads. The application threads call umap to map in one or more virtual address ranges, using the same API as mmap and can optionally pass in file and offset information. As an extension to the mmap API, umap can accept a list of files and offsets. Additionally UMap has a plugin architecture, allowing application threads to register callback functions to process the read or write page faults. Once, a UMap handler has been started, application threads can then access locations in the memory mapped range, triggering page faults when a page is not resident. As shown by the dotted lines in Figure 1, a UMap handler thread is notified when a page fault to the handler's registered range occurs. The handler thread satisfies the page fault by calling the application-supplied function if provided, and otherwise performing the default action of direct I/O to the backing file (typically on node-local SSD), using the UFFDIO_COPY ioctl to ensure atomic copy to the allocated memory page.

UMap can be configured with an application-specific page buffer size, allowing the application to control the amount of memory consumed in resident pages. The handler maintains minimal metadata: a simple circular buffer tracks pages faulted

in and their modification state. When the buffer is full, the next page in the buffer is evicted, implementing a FIFO replacement policy.

UMap can be built in two different userfaultfd modes. Read-only mode is used for applications that map in existing data sets and don't modify them. This mode is suitable for analyzing large data sets residing on SSD and is supported in released Linux kernels 4.3+. Read-write mode is used for applications that update locations in the mapped range. For example, an in-place sort modifies the memory regions as buckets are moved. For read-write mode, a custom kernel is required, as the write userfaultfd extensions have not yet been mainlined. Userfaultfd read-write mode is available in a set of kernel patches maintained by the userfaultfd developer [6].

IV. Application-specific page sizes

A UMap handler can be configured to use a page size that is a multiple of the system page size. To service the fault, the large page is read or written in a single I/O request. Figure 2 shows an experiment with Breadth First Search of a scale 28 power law graph in which the page size is successively doubled six times with two different size page buffers. Both trend lines show that BFS benefits from larger page size, with the amount of performance improvement diminishing past 32K for the 32GB buffer size.

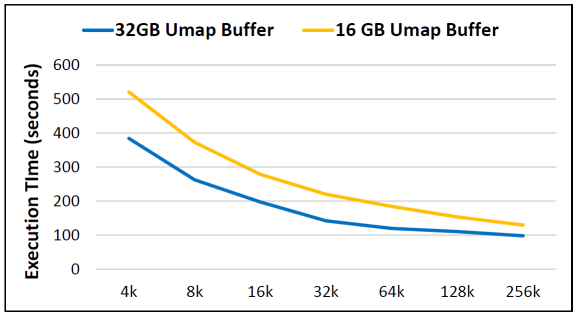

Fig. 2: Execution time scaling with page size

V. Materializing 3D image cubes

As a user space handler, UMap can easily be extended to support complex applications. An on-going study of telescope data sets to search for transient objects such as asteroids uses UMap to track potential transients in time series imagery that is abstracted as a 3D image cube. The data is contained in 10's to 100's of FITS format image files. The application maps the "cube" into a virtual address range. Page faults are handled by the FITS plugin callback function supplied to the UMap handler. The handler is initialized with a list of files, and the FITS plugin parses the file headers to locate offset of the image array within each file. The plugin handles a page fault by locating the desired page at the correct offset in one of the files and reading it into memory with a direct IO request. The read may span more than one file.

The application simply loops over the cube i.e. Cube[k][i][j], where k sweeps the time dimension and i and j traverse a single image. The transient detection algorithm computes the pixel-wise median of a vector tracing a potential path of a transient object. The outermost loop processes a list of vectors, as shown in Figure 3. For this use case, we have demonstrated processing data from the LSST DC1 synthetic data collect (with transients artificially injected), and are evaluating algorithms to generate plausible vectors along which to compute the median. For blind search, random vectors are used.

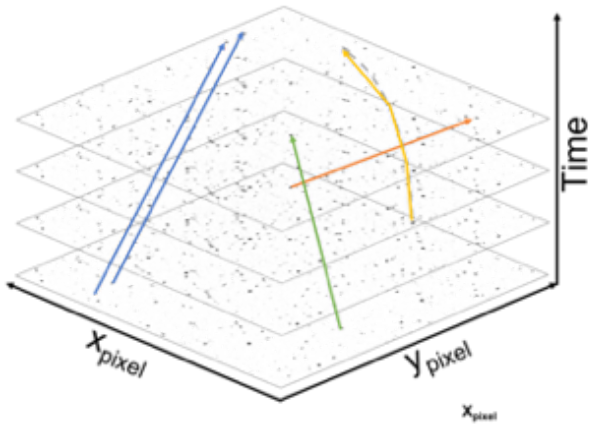

Fig. 3: Materialized 3D image cube

VI. Conclusions

Memory-mapped I/O provides a convenient mechanism for analyzing out-of-core data sets. The highly configurable user space UMap handler gives applications control over many aspects of memory mapping while confining those customizations entirely within the application itself. The application can control the page buffer size, the page size, and even how the page fault is resolved. UMap's customization allows the programmer to tune performance and also provide functionality not easily available in a simple file-backed mmap().

Acknowledgments We thank Xiao Liu and Jishen Zhao (then associated with UC Santa Cruz) for their participation in an early phase of the project. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-PRES-746244). Funding provided by DOE ECP Software Technologies program.

References

[1] B. Van Essen, H. Hsieh, S. Ames, R. Pearce, and M. Gokhale, "DI-MMAP–a scalable memory-map runtime for out-of-core data-intensive applications," *Cluster Computing*, 2013.

[2] M. Jiang, B. Van Essen, C. Harrison, and M. Gokhale, "Multi-threaded streamline tracing for data-intensive architectures," in *IEEE Symposium on Large Data Analysis and Visualization*. IEEE, November 2014.

[3] "Userfaultfd." [Online]. Available: https://www.kernel.org/doc/Documentation/vm/userfaultfd.txt

[4] A. Arcangeli, "Userland page faults and beyond why how and what's next." [Online]. Available: https://schd.ws/hosted_files/lcccna2016/c4/userfaultfd.pdf

[5] B. Caldwell, Y. Im, S. Ha, R. Han, and E. Keller, "Fluidmem: Memory as a service for the datacenter," *CoRR*, vol. abs/1707.07780, 2017. [Online]. Available: http://arxiv.org/abs/1707.07780

[6] A. Arcangeli, "aa.git." [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/andrea/aa.git