NVM Allocator in Disaggregation Era

Teng Ma[1, *], MingXing Zhang[1, 3], Dongbiao He[1, *], Kang Chen[1], and Yongwei Wu[1]

[1]Department of Computer Science, Tsinghua University, [2]Sangfor, [*]Student, Email: {*mt16, hdb13*}*@mails.tsinghua.edu.cn*

Recently, a hot trending of rack-scale architecture named disaggregation was proposed to strength the data center. In industry, Intel "RSD" and HP "the machine" will soon be available as real-world DC resolutions. Some researches like Infiniswap [3] and Zombieland [4] aim to find out the correct deployed guidelines and how to optimize disaggregation architecture. Disaggregation architecture divides machines into working one and resource one (i.e., blade server). Infiniswap adopts network swapping method to map the memory resource and hence organizes memory as a block pool. Zombieland mainly proposes a new power state to reduce the resource footprint and uses one-sided RPC allocator. To best of our knowledge, except these two works, other disaggregation works lack discussion about resource allocator design. In the future, with the lower latency RDMA, NVM will also be deployed in the blade server [1, 2]. However, similar as other disaggregation architecture, NVM disaggregation also has a requirement of the resource management. Moreover, in a distributed environment, keeping allocation status consistency and persistence will be harder, as well as bringing a huge overhead in the network. Obviously, previous methods cannot fit the NVM disaggregation well since lacking considerations of byte-addressable, wear leveling and other NVM features. So we need to exploit the non-volatile of disaggregation NVM with the volatile working nodes and adopt suitable designs to manage the disaggregation NVM.

In a disaggregation scenario, to build up a best-effect NVM allocator, we face four challenges: 1) How to design the protocol between blade server and working node to allocate NVM with high-speed RDMA; 2) How to guarantee the reliability of allocator (i.e., recovery, crash consistency and stable allocation status); 3) How to reduce the network overhead and reach a higher performance; 4) How to improve the utilization of NVM.

We propose a design paradigm named RAllocator which decouples different functions to working nodes and blade servers. The blade servers are responsible for handling the failure, and the working server will manage resource with a fined grained respectively. Due to the frequency allocation requests from the working node, RAllocator adopts a two-tier architecture to release the stress in the blade server (it always has lower power). It applies a slab-based grained in the back-end and only provides several fixed-size slabs to the working node. The working node can request slab from the blade server and take control of slabs with fine-grained resource management.

We implement a one-sided RDMA protocol to release blade server from handling network (i.e., bypass CPU) and hence reduce the network overhead. In RAllocator, blade server will pre-allocate enough slabs and provide the NVM address to working nodes. To support concurrent, there is a 64-bits sequencer in the blade server, and the working node first fetches a sequencer number via `RDMA_fetch_and_add` (atomic verbs). Notice that the sequencer in the blade server can sequence the allocation request from working nodes. Then the working node uses this sequence number as offset to fetch back the resource address and returns a acknowledge to hint blade server the completion. When freeing a slab, the working node writes a request to the blade server's buffer and then it uses epoch-based method to reclaim the resource.

The working node organizes slabs as three full/partial/empty list according to its usage status. When the status of a slab turns from full/partial to empty, it won't be immediately free but push to the empty list for reuse. Thus, a threshold of empty slabs limitation is set to eliminate lower utilization of slabs in the working node.

In the blade server, its equipped NVM is responsible for persisting allocation status. There is an allocation bitmaps stored in NVM to indicate whether a slab is allocated. Besides, several critical metadata of the allocator are stored in a "well-known" location (a.k.a., global naming) for fast recovery. We design a mechanism to assist working node to recover allocation status from persistent bitmaps and metadata in the blade server.

According to the different workload, RAllocator uses a tuning strategy to automatically choose the slab size in the running time to improve the resource utilization. With multiple blade servers, a coordinator is necessary to make load shedding and avoid resource overused. This coordinator can monitor the resource utilization of every single blade server and hint the working node to choose the appropriate blade server. In our future plan, the blade server will support wear leveling and defragmentation to amplify the advantages of disaggregation. Moreover, we will evaluate DAllocator with end-to-end applications.

References

[1] Nvmf based integration of non-volatile memory in a distributed system. `https://www.openfabrics.org/images/2018workshop/presentations/111_BMetzler_NVMfLessons.pdf`.

[2] Openfam api: programming model for disaggregated persistent memory. `http://storageconference.us/2018/Presentations/Keeton.pdf`.

[3] Gu, J., Lee, Y., Zhang, Y., Chowdhury, M., and Shin, K. G. Efficient memory disaggregation with infiniswap. In *NSDI* (2017), pp. 649–667.

[4] Nitu, V., Teabe, B., Tchana, A., Isci, C., and Hagimont, D. Welcome to zombieland: practical and energy-efficient memory disaggregation in a datacenter. In *EuroSys* (2018), ACM, p. 16.