# Summary of Efficient Memory Disaggregation with Infiniswap

Vandit Agarwal, Chun-Jung Chiu, Ting-Wei Cho

**Problem:**

Memory-intensive applications suffer significant performance deterioration when their working sets do not fit in memory. One of the causes of this major performance loss is because disks are orders of magnitude slower than memory, and servers “page out” to disks when they run out of memory. Although memory disaggregation in an RDMA-enabled cluster is a possible solution, where servers page out to available memory of other machines instead of local disks, current proposals are not feasible since they require new hardware designs and programming models. In addition, current proposals/systems have limited scalability due to requirements for central coordination to monitor memory availability and paging activities, and remote CPU usage for data transmission (block devices such as nbdX).

**Hypothesis:**

This paper made hypotheses as follows:

- Imbalance in memory utilization is present across cluster nodes
- Memory utilization at individual nodes are relatively stable in a short period of time
- More and more applications are deployed using container-based model
- Remote evictions and failures are common

**Solution:**

I. **How to make applications whose working memory do not fit in memory faster by utilizing available memory across the cluster?**

*Network paging and memory disaggregation:* Main memory pages out to remote, underutilized memory instead of local disks. This idea becomes feasible as RDMA begins to be widely available.

II. **How to take advantage of RDMA clusters without requiring major modifications in infrastructures/OSes/applications?**

*A conventional block device I/O interface:* Infiniswap serves as a swap space that exposes a conventional block device I/O interface to the VMM of the OS.

III. **How to ensure scalability and fault tolerance?**

a. **Scalability:**

Infiniswap adopts decentralized design to avoid the need of a central coordinator, which may be a bottleneck when the size of the cluster increases. Machines can work together through Infiniswap daemon and block device installed on each machine. Also with RDMA protocol, Infiniswap does not become CPU bound as other TCP/IP systems.

b. **Fault tolerance:**

There is no single point of failure for Infiniswap thanks to its decentralized design. If a remote machine fails, Infiniswap relies on other remaining remote memory and local backup disk. For each slab placed synchronously on remote memory, there is a copy written asynchronously on local disk.

IV. **How to ensure load balance and minimize the influence of remote evictions/failures on local application?**

The entire address space of an Infiniswap block device is divided into multiple ***slabs*** of fixed size to be the unit of remote mapping and load balancing. A slab starts out in the *unmapped* state, and is monitored for its page activity using EWMA with one second period. If a slab's page activity exceeds a set threshold, remote placement will be initiated to map it to a remote machine's memory. A bitmap is used to maintain the remote locations of pages written out to remote machines. If slabs of a block device are well distributed across the cluster, both load balancing and minimum influence of evictions/failures can be satisfied.

V. **How to well distribute slabs across the cluster with minimum communications?**

***Power of two choices*** is used to ensure slabs are mapped to as many remote machines as possible, where two of the machines without slabs from this block device are contacted, and the one with more available memory is selected.

VI. **How does Infiniswap block device handle I/O requests from CPU cores?**
Each core is configured with a software staging queue to stage requests, which are placed on disk/RDMA dispatch queues and forwarded to disk and/or remote memory after consulting page bitmap. For page write requests, if the corresponding slab is mapped, the requests are duplicated and put into both disk and RDMA dispatch queues for fault tolerance. If the slab is unmapped, the request will be sent to disk dispatch queue only. For page read request, if a slab is mapped, the request will be put into RDMA dispatch queue. Otherwise, Infiniswap reads from disk.

VII. **How does Infiniswap block device handle remote evictions and failures?**
If a remote daemon sends *EVICT* message or becomes unavailable to the block device, the corresponding slabs are marked *unmapped* and portion of bitmap are reset. All future requests go to disk. In eviction scenario, the block device will wait until RDMA dispatch queue is complete before responding to the remote daemon with *DONE* message. In failure scenario, especially in *read-after-write* scenario, where the remote memory is written but not the disk, and the VMM request to page in the page, Infiniswap can direct the read request to disk dispatch queue since the previous write request has already been in the queue ahead of the read request. This way the updated version will be paged in the main memory.

VIII. **How does Infiniswap make sure there's enough memory for local applications?**
The Infiniswap daemons monitor memory using EWMA similar to the way to monitor page activities of a slab. If the available memory is higher than a set threshold, the daemon proactively allocate *unmapped* slabs to save initialization overhead. If on the other hand, the available memory is less than the set threshold, the daemon proactively evict mapped slabs.

IX. **How to avoid evicting very active slabs efficiently with minimum communications?**
If the daemon chooses to evict slabs randomly, chances are high that it will evict very busy slabs, making the performance of applications running on machines that remote page to this machine suffer. Checking the activeness of all the slabs would incur high communication overhead although it guarantees to find the least active slabs. By leveraging ***generalized power of choices***, where the activeness of E + E' slabs are checked (E is the total number of slabs to be evicted, and E' can be any number as long as E' <= E), among which E least active ones are selected to be evicted, the probability of active slabs being evicted decreases significantly.

**Limitations and Improvements:**

- INFINISWAP makes use of moderately large SlabSizes. Though this reduces metadata management overhead, too big a SlabSize might make it less flexible and at the same time decrease the space efficiency of a remote memory. Automatically finding the optimal SlabSize is a possible future improvement to consider.
- Application transparency limits INFINISWAP's performance for various applications.
- Currently INFINISWAP works independently of the operating system as a result of which it cannot provide for predictable performance. It has to have some sort of controlled swap mechanism inside the operating system to provide the required performance predictions.
- Currently INFINISWAP is not able to distinguish between different applications. Being able to do this, however, would lead to much better resource management and application sandboxing.
- Network bandwidth is a major bottleneck in any data center operation. This has historically led to concepts such as localisation etc. INFINISWAP assumes that RDMA connectivity would not be a bottleneck, but this may not be a reasonable assumption in the future to make. The performance of the whole system primarily banks on how fast the network is and in future if the network becomes a bottleneck then there is no

perceivable advantage of using the proposed solution anymore. Professor mentioned in the lecture that using erasure coding instead of replication can ease the pain from network bandwidth limitation.

**Summary of Class Discussion:**

- Why change in performance is not as significant when moving from 100% workload to 75% workload while it is very significant when moving from 75% to 50%?
  The memory utilization varies over time. For graphX and powergraph, the peak usage is high but the average usage is stable and low. Going from 100% to 75% did not have much effect because the application peaked only for a short period of time while remaining relatively stable for the most of the time. When it was at 50%, most of the runtime was affected.
- In figure 2 of the paper, the Y axis is the cumulative distribution function.
  We want the median machine and the machine at the tail to perform very close to each other. In most cases, however, they are pretty far away from each other. Almost half of the time, they are almost 2x times worse. More than half of the time, about half of the memory is not utilized.
- Who is responsible for setting the threshold?
  If the threshold is too low, then it will be mapped but we are not using it and it gets to waste. If it is set too high then that means that it has already hit the disk too many times and there is no benefit there. Currently, it's very hard to determine the sweet spot where we can say with high confidence the time right before the application peaks. In such a situation, generally in systems research, the users are given a configurable knob. In general, the more knobs a system has, the worse designed the system is.
- Why do we need slabs in terms of scalability?
  It reduces communication. We have to register and deregister memory before and after RDMA. There is a cost associated with it. The cost is low but it is not negligible when we start scaling up. Also, with this, the amount of bookkeeping is reduced as everything that happens is restrained within the slab itself.
- Better way to find machines that are free:
  One way to find the most free machine is to scan through the entire system and look for the most free machine. The other way is to randomly pick up any 2 machines and choose the one that is more free of the two. It turns out that the latter results in similar results as the first one with very low overhead
- How would one avoid going to the disk even if something has failed?
  The answer is in replication. We need to find a balance in this scenario though to find the right kind of replication.
- What will happen if we send 2 copies while paging out, and before it can be written to disk it is buffered?
  It is going to block. We cannot throw it out until it is not written to the disk. If a new page out occurs, we cannot do a remote write and tell the kernel that we are done as we cannot write to buffer and it is going to block. It primarily depends on the size of the buffer (default size is 1GB). Buffer drain speed is restricted to the speed of the disk. That is the reason that we want to avoid using disks completely. Also, finding optimal buffer size is an issue, we do not know about the size of the burst that an application will have.
- New concept: serverless:
  It is a cool new concept that is cheap. The drawback though is that it is stateless. To make it stateful, we need to write to a persistent disk, which is again slow.
- Method to counter replication:
  Erasure coding involves only about 25% overhead as compared to a 100% overhead in case of replication. In erasure encoding, the amount of data that we are reading is k times smaller if the data is split into k smaller pieces. If 1 of the machines is slow, then only reading from 1 of the replicas will be slower and the overall application would not suffer. In case of replication, in case the machine where the replica is stored is slow, then the entire read becomes slow. Although erasure coding requires time for CPU computing, it is less than the time saved by adopting this approach.