Efficient Memory Disaggregation with InfiniSwap

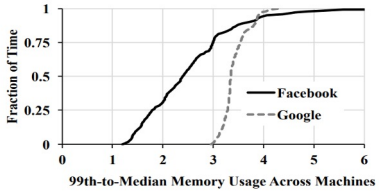
Divy Patel (dspatel6@wisc.edu)

Challenges

- Memory-intensive apps are usually low latency and high throughput due to their access patterns
- But see performance loss, when working sets do not fully fit in the memory
  - Potential Mitigation: Right-sizing memory allocation, but leads to under-utilization and unbalanced memory usage across the cluster
- Cannot leverage under-utilized remote memory when paging out to disks
- Existing memory disaggregation solutions requires change in:
  - Operating System
  - Infrastructure
  - Applications

Characteristics of Memory Imbalance (1 of 2)

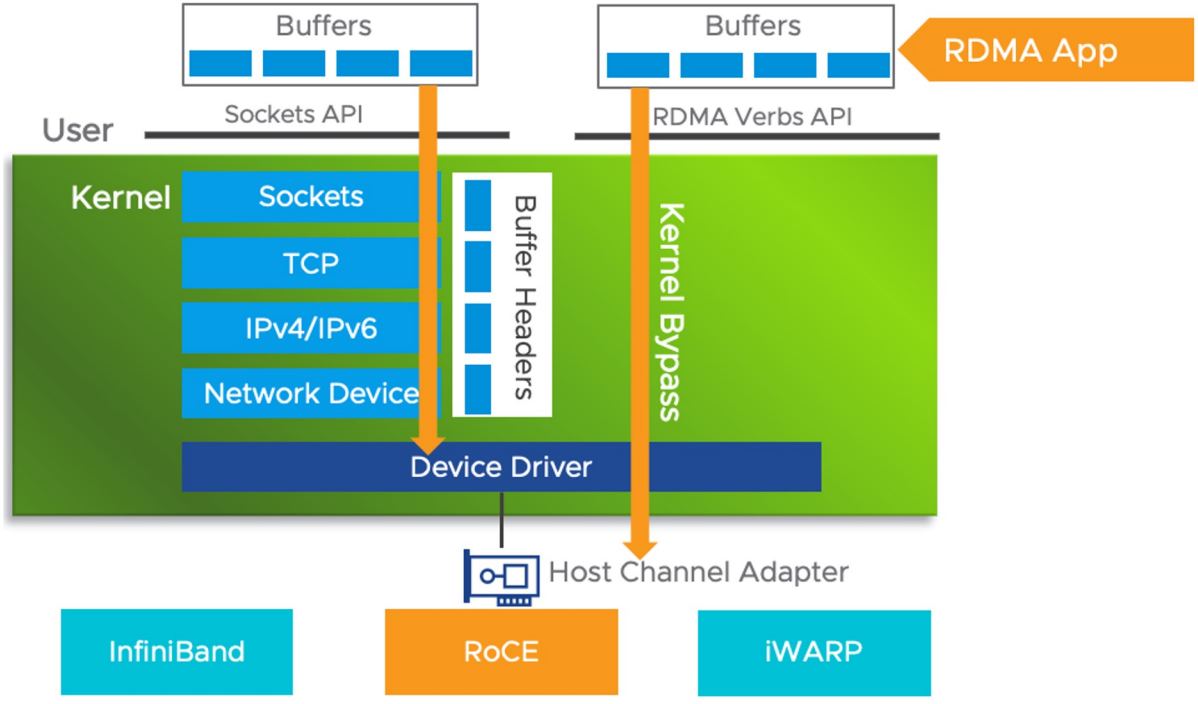
- Presence of Imbalance
  - Memory usage across machines can be substantially unbalanced in the short term(within 10 seconds)
  - Memory utilization imbalance is measured by calculating the 99th-percentile to the median usage ratio over 10-seconds intervals
  - In ideal case, this should be 1 for most of the time
  - However, from production traces, it was 2.4 in Facebook and 3.35 in Google more than half the time
  - Implies, most of the time, more than a half of the cluster's aggregate memory remains unutilized.



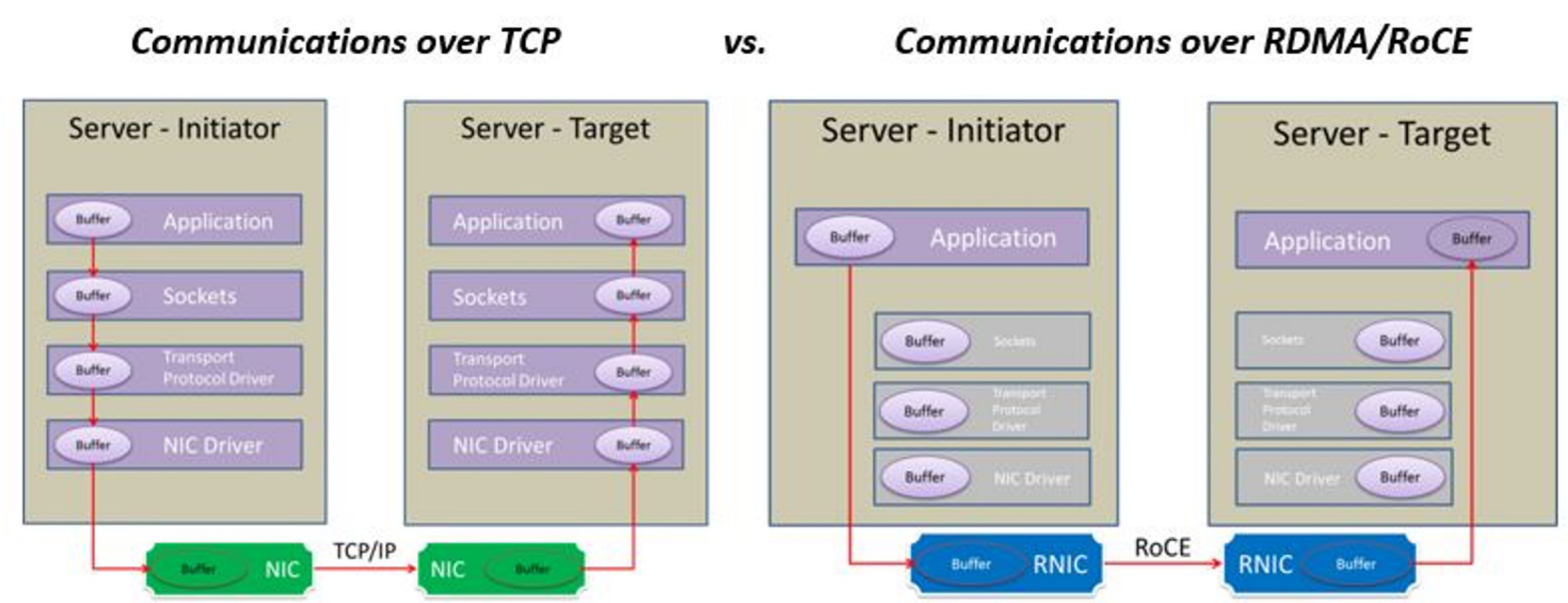
Characteristics of Memory Imbalance (2 of 2)

- Temporal Variabilities
  - Although skewed, memory utilizations remained stable over short intervals
  - Ut(m) at time t to be stable for the duration T if the difference between Ut(m) and the average value of Ut(m) over the duration T remains within 10% of Ut(m)
  - For most unpredictable machine in the Facebook cluster, the probabilities of Ut(m) being stable for the next 10, 20, and 40 seconds were 0.74, 0.58, and 0.42, respectively
  - For Google, the corresponding numbers were 0.97, 0.94, and 0.89, respectively
  - Higher probabilities in the Google cluster are due to its long-running services, whereas the Facebook cluster runs data analytics with many short tasks

RDMA(Remote Direct Memory Access) (1 of 3)

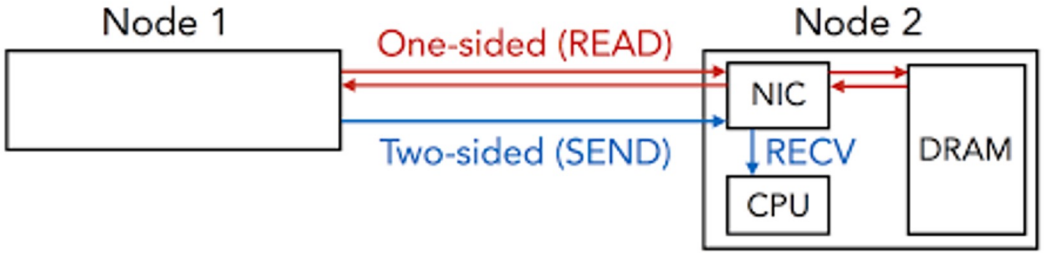


RDMA (2 of 3)

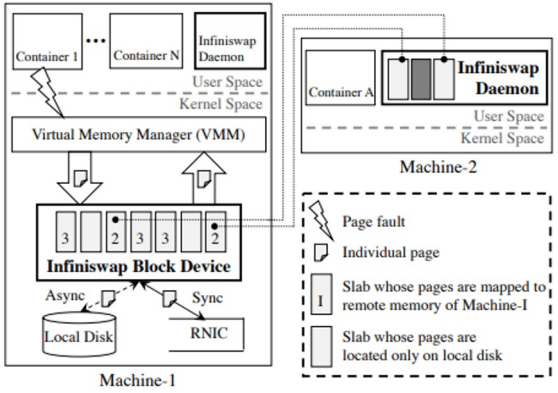

Communications over TCP
vs.
Communications over RDMA/RoCE
Server - Initiator
Server - Target
Application
Sockets
Transport Protocol Driver
NIC Driver
Buffer
NIC
TCP/IP
Server - Initiator
Server - Target
Application
Sockets
Transport Protocol Driver
NIC Driver
Buffer
RNIC
RoCE

RDMA (3 of 3)

- Analogous to REST API verbs => get, put, post
- One-sided RDMA verbs
  - **READ** and **WRITE**
  - Bypasses remote CPU, hence, low latency and high throughput
- Two-sided RDMA verbs
  - **SEND** and **RECV**
  - Used for cases where the nodes require sync



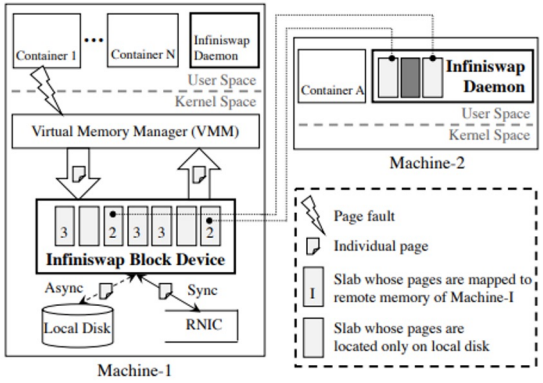
Architecture (1 of 2)



- Two primary components:
  - Infiniswap Daemon
  - Infiniswap Block Device
- Block device exposes a conventional block device I/O interface to the virtual memory manager
- Address space is logically partitioned into fixed-size slabs
- Slab is the unit of remote mapping and load balancing
- On the daemon side, a slab is a physical memory chunk of SlabSize that is mapped to and used by an block device as remote memory
- Slabs from the same device can be mapped to multiple remote machines' memory for performance and load balancing

Architecture (2 of 2)

- For page-out requests, if a slab is mapped to remote memory
  - sync to remote memory using RDMA WRITE
  - async to the local disk
- If it is not mapped
  - sync only to the local disk
- For page-in requests or reads
  - consult the slab mapping
  - read from remote memory using RDMA READ
- Daemon responds to slab-mapping requests of block device
- Also, pre allocates its local memory to minimize time overheads
- Proactively evicts slabs, when necessary, to minimize impacts on local applications
- Control plane communications take place using RDMA SEND/RECV.

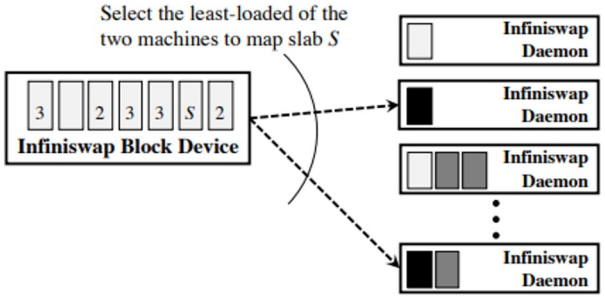


Slab Management

$$A_{\text{current}}(s) = \alpha\, A_{\text{measured}}(s) + (1-\alpha)\, A_{\text{old}}(s)$$

- Logical division of its entire address space into multiple slabs of fixed size(SlabSize), simplifies slab placement and eviction algorithms
- Each slab starts in the unmapped state
- Monitoring the page activity rates of each slab using an exponentially weighted moving average (EWMA) with one second period
- When the rate crosses a threshold (HotSlab), remote placement is initiated
- HotSlab => 20 page I/O requests/second
- To keep track of location, it maintains a bitmap of all pages
- All bits are zero. If a page is written out to remote memory => corresponding bit is set
- Remove a slab from remote memory if rate goes below a ColdSlab threshold

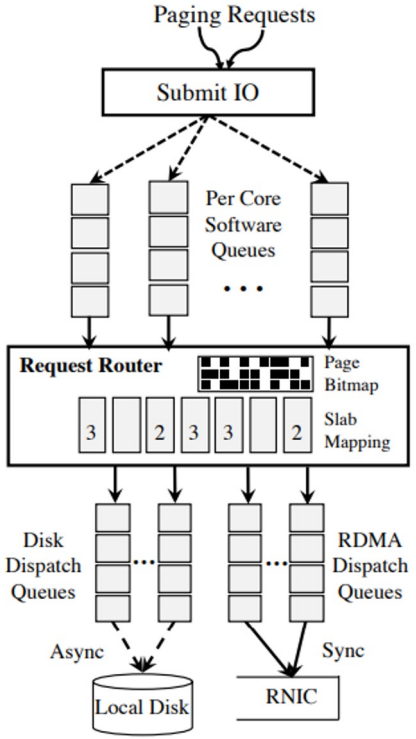
Remote Slab Placement



- Goal: Must distribute slabs from the same block device across as many remote machines as possible. To minimize the impacts of future evictions/failures
- It is decentralized to provide low-latency mapping without central coordination
- Leverages **power of two choices,** when slab is marked as HotSlab
- Divides all the machines into two sets: those who already have any slab of this block device (M_{old}) and those who do not (M_{new})
- From M_{new}, it contacts two daemons and selects the one with the lowest memory usage

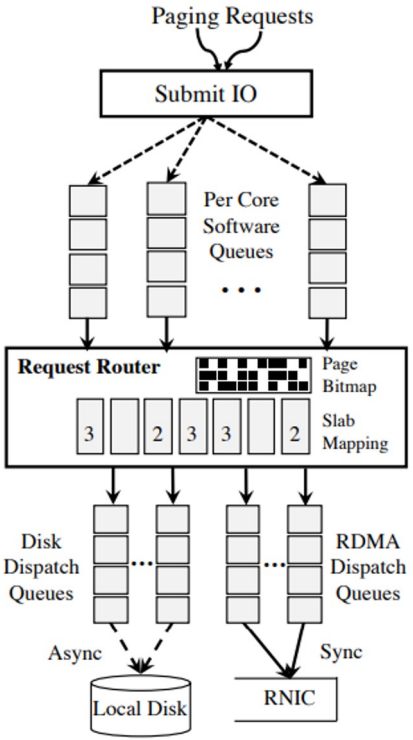
I/O Pipelines (1 of 3)

- Each CPU core has a staging queue, where block(page) requests are staged
- Request router consults the slab mapping and the page bitmap to determine how to forward them to disk and/or remote memory
- Number of RDMA dispatch queues is the same as that of CPU cores
- Each request is assigned to a random dispatch queue by hashing to load balance



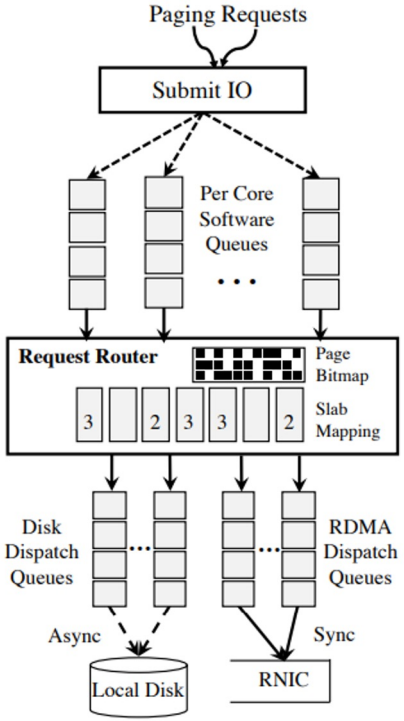
I/O Pipelines: Page-Write

- Page write: if slab is mapped, put into both RDMA and disk dispatch queues
- Content is copied into RDMA dispatch entry, and shared between the requests
- After RDMA WRITE completes, the page write is completed and its physical memory is reclaimed by the kernel without waiting for the disk write
- RDMA dispatch entry and its buffer will not be released until the completion of the disk write operation
- For unmapped slabs, only put into the disk dispatch queue



I/O Pipelines: Page-Read

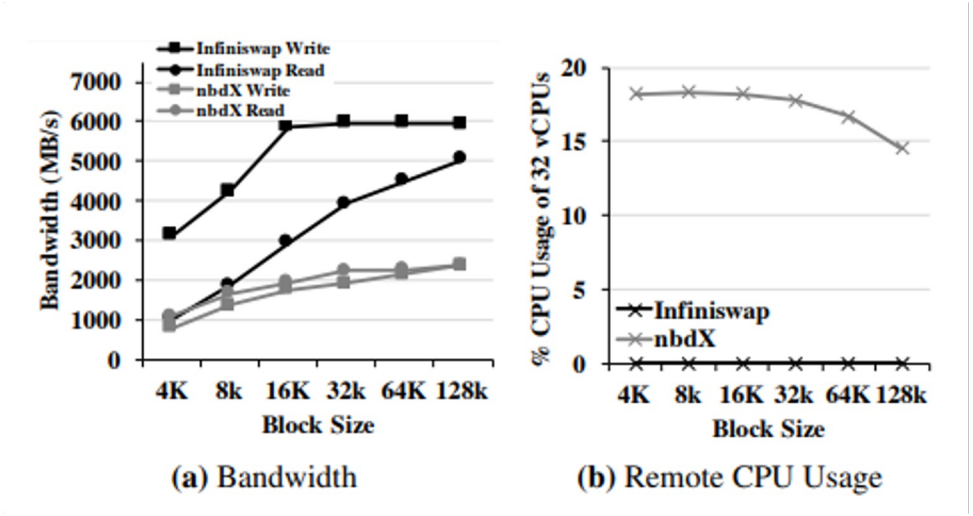
- Page Reads: if the slab is mapped and the page bitmap is set, an RDMA READ operation is put into the RDMA dispatch queue
- When the RDMA READ completes, page-in is completed
- Otherwise, reads it from the disk.



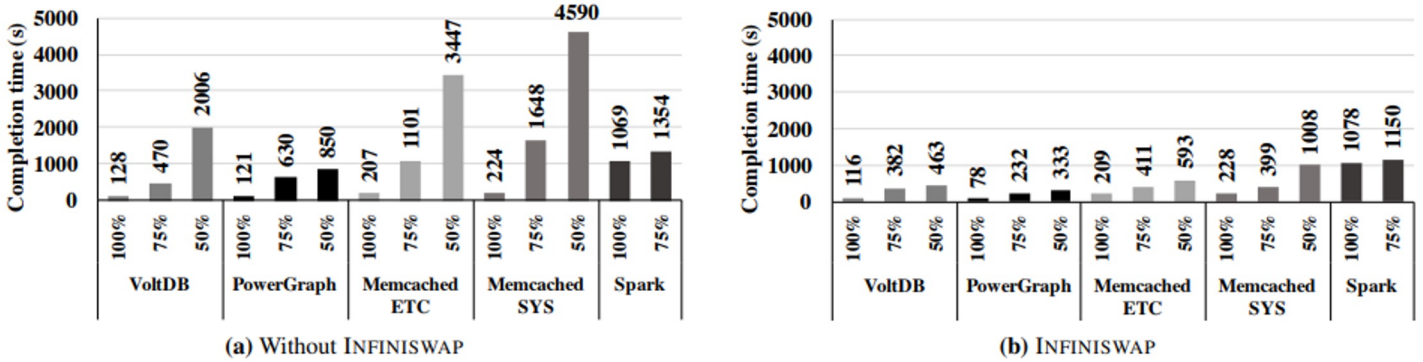
Handling Slab Evictions/Remote Failures

- **Remote Eviction:** Decision to evict a slab is communicated to a block device via the EVICT message from the corresponding INFINISWAP daemon
- Upon receiving this message, the block device marks the slab as unmapped and resets the corresponding portion of the bitmap. All future requests will go to disk
- **Remote Failures:** If the requests to remote memory do not complete, this is considered to be as remote failure, and the corresponding slab is marked as unmapped and bitmaps are reset to 0
- In the current implementation, INFINISWAP does not handle transient failures separately. A possible optimization would be to use a timeout before marking the corresponding slabs unmapped

Evaluation: Performance as Block Device



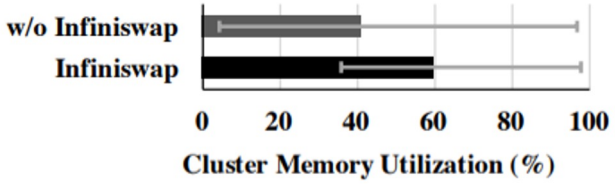
Evaluation: Impact on Applications



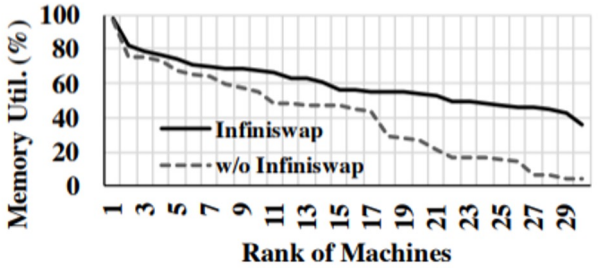
(a) Without INFINISWAP

(b) INFINISWAP

Evaluation: Cluster-wide Performance



(a) Cluster memory utilization



(b) Memory utilization of individual machines

Thank you