Md Majid Jahangir | SBU ID - 112077145
AMS 560 - Big Data Systems, Algorithms and Networks

Assignment 3 (Paper Reading)
Efficient Memory Disaggregation with INFINISWAP

Motivation : Nowadays, memory intensive applications such as Redis, Memcached, Spark, etc, are being used widely for services which needs low latency high throughput as requests served from memory are faster than from disk access. However, when the entire working set of pages for an application cannot fully fit in memory, the performance of such applications deteriorate as these applications start paging and swapping pages in and out of swap devices to physical memory ultimately leading to a phenomena called thrashing. Even though this problem can be solved by either increasing the memory capacity of each machine or by correctly predicting the right size of memory allocation, these measures could cause an imbalance as applications often overestimate their memory requirements resulting in severe underutilization and unbalanced memory usage across the entire cluster. Also, many existing proposals for memory disaggregation are possible only with new architectures or new hardware designs and new programming models that makes them infeasible. Hence, this paper proposes an efficient memory disaggregation solution called InfiniSwap which does not require modifications to the infrastructure, operating systems or applications.

Prior Solutions: Some of the prior works on memory disaggregation discussed or mentioned on the paper are - “Network Support for Resource Disaggregation in Next-Generation Datacenters” [2] which talks about how a datacenter has been architected as a collection of servers where each server aggregates a fixed amount of memory, storage, computing, etc, and how efficiency and performance could be improved if the resources within a server are disaggregated and data center is instead architected as a collection of standalone resources. Hence, this paper talks about the feasibility of building a network that enables disaggregation at datacenter scales; however this paper calls for a new architecture design altogether.
Another such existing solution - “Disaggregated Memory for Expansion and Sharing in Blade Servers” [3], which proposes a general purpose architectural building block called a memory blade that will allow memory to be disaggregated across a system and it can be used for memory capacity expansion and for sharing memory across servers to reduce provisioning and cost and improve performance; however it involves a new hardware design.
One of the prior works done on remote paging model is - “A Transparent Remote Paging Model for Virtual Machines” [4] which proposes a hypervisor based remote paging in virtual machine systems allowing a virtual machine to transparently use the memory resource on other physical machine as a cache between its virtual memory and virtual disk device. The goal was to alleviate the impact of thrashing by reducing the average disk I/O latency.

Key Ideas of Proposed Solution: The key ideas of the proposed solution are -

1. InfiniSwap is a decentralised memory disaggregation solution for RDMA clusters that efficiently exposes all of the cluster’s memory to user applications without any modifications to the applications or operating systems of the individual machines; and uses remote memory for paging.
2. It consists of two primary components -
    a. InfiniSwap block device which exposes a block device I/O interface to the virtual memory manager (VMM) of OS. VMM treats this as a fixed size swap device and the address space of it is partitioned into fixed size slabs. A slab over here is a unit of remote mapping and unit of load balancing in InfiniSwap.
    b. InfiniSwap daemon - which runs in the user space and participates in control plane activities namely responding to slab-mapping requests from InfiniSwap block devices; preallocating its local memory to minimise time overheads in slab-mapping initialization and in evicting slabs when necessary ensuring minimum impact on local user applications. Over here, a slab is a physical

memory that is mapped to and used by an InfiniSwap block device as remote memory.

3. InfiniSwap has been implemented as a loadable kernel module for Linux.
4. Since it does not have a central coordinator and is decentralised, it doesn't have a single point of failure and hence is fault tolerant. If a remote machine fails, InfiniSwap relies on other remote machines.
5. A slab always starts in an unmapped state; and InfiniSwap monitors the page activity rates of each slab using an exponentially weighted moving average (EWMA) and whenever the page activate rate of a slab crosses a threshold, InifiniSwap initiates a request to map the slab remotely. Whenever a slab is mapped to a remote memory, a write request will be put into disk dispatch queues and RDMA queues; and if it is not mapped then it is just put into the disk dispatch queue. And, for any page-in requests or reads, InfiniSwap will lookup the slab mapping in order to read from the appropriate source; InfiniSwap maintains a bitmap of all pages to keep track of mapped pages. When a slab is not mapped its bit is set to 0, and when it's mapped to a remote memory, its corresponding bit in the bitmap is set; also on slab eviction or remote machine failure in which the slab was mapped, the bit is reset.
6. The algorithm to remotely place the slab ensures that the slabs are distributed uniformly across as many remote machines to minimize the impact of future eviction from remote machines; and also to balance memory utilization across all machines minimizing the probability of future evictions. In order to achieve this, InfiniSwap divides all of the remote machines in two sets - ones which already contains the slab of that block device and those who does not have any slabs of that block device. Then it first selects a machine from latter set and contacts its daemon to find its memory usage and then a machine from the former set and its daemon and then chooses the one which has the lowest memory usage thus balancing the load in decentralised manner.
7. For a little more details on key ideas of solution, refer to "How do the two components of INFINISWAP work?" section which explains further how the two components interact.

Performance: The performance of InfiniSwap was evaluated on a 32 machine, 56 Gbps Infiniband cluster on CLoudLab. InifiniSwap provides 2-4 times higher I/O bandwidth than Mellanox and nbdX. InifiniSwap did not saturate any remote virtual cores while nbdX saturated 6 remote virtual cores. On running memory intensive applications like VoltDB, Redis, Memcached, Apache Spark, etc, InfiniSwap improves the throughput by upto 4 times 0.94 times to 15.4 times 7.8 times over nbdX disk and tail latencies by upto 61 times 2.3 times, by without modifying any of the above applications. It was also observed that on remote machines failure and evictions InfiniSwap ensures fast recovery with very little impact on remote applications. InfiniSwap also improves cluster memory utilization by 1.47 times through its decentralised setting by using a very small amount of network bandwidth.

Difference between INFINISWAP and other solutions for memory aggregation? [Key Question]:

1. "Network Support for Resource Disaggregation in Next-Generation Datacenters" [2] which talks about how a datacenter has been architected as a collection of servers where each server aggregates a fixed amount of memory, storage, computing, etc, and how efficiency and performance could be improved if the resources within a server are disaggregated and data center is instead architected as a collection of standalone resources. Hence, this paper talks about the feasibility of building a network that enables disaggregation at datacenter scales; however this paper calls for a new architecture design altogether. On the contrary, InfiniSwap also implements an efficient memory disaggregation algorithm but does not call for a new architecture design and can be used in the current same architecture of datacenters as it is implemented as a loadable kernel module in Linux.
2. "Disaggregated Memory for Expansion and Sharing in Blade Servers" [3], which proposes a general purpose architectural building block called a memory blade that will

allow memory to be disaggregated across a system and it can be used for memory capacity expansion and for sharing memory across servers to reduce provisioning and cost and improve performance; however it involves a new hardware design. On the contrary, InfiniSwap is implemented as a loadable kernel module which provides a two primary components - block device, which could be mounted on any existing hardware, without any need for any new hardware designs or modification to existing hardware; and a daemon which runs on user space to monitor local memory usage.

3. “Accelerating Relational Databases by Leveraging Remote Memory and RDMA” [5], which proposes a method of abstracting remote memory accessed via RDMA, using a lightweight file API to solve how an SMP RDBMS can leverage RDMA and available remote memory in cluster and hence improving performance of memory-intensive workloads, however, they use a centralized method of achieving this. On the contrary, InfiniSwap is a decentralised memory disaggregation service which achieve the same without any central coordination between remote machines and hence improves performances and throughputs even further.

How is the performance measured? [Key Question]: The performance is measured in four different aspects -

1. Performance by just considering the InfiniSwap block device - The InfiniSwap block device was compared against nbdX by using a well known disk benchmarking tool called fio. For both of them , parameter sweeps were performed by varying number of threads in fio from 1 to 32 (since 32 machines were used). In terms of bandwidth InfiniSwap performs between 2 times and 4 times better than nbdX. It was found that nbdX had excessive CPU overheads since it would copy data to and from RAMdisk at remote side and hence it became CPU bound for smaller block sizes and would also often saturate 6 virtual cores it would run on. On the contrary, InfiniSwap would bypass remote CPU and has almost zero CPU overheads in remote machine.
2. Performance on multiple memory intensive applications - Single machine was considered with three configurations of 100%, 75% and 50% of the peak memory usage for each application. 4 memory intensive applications and varying workloads was used like -
    a. TPC-C benchmark on VoltDB - TPC-C performed 5 different types of transactions on VoltDB. 8 sites in VoltDB was set to achieve a single container workload of 11.5 GB and 2 million transactions. It was observed that performance using InfiniSwap dropped linearly instead of super linearly when smaller amounts of workload fit in memory. With InfiniSwap, VoltDB experienced only 1.5 times reduction in throughput instead of 24 times using disk. InfiniSwap also improved VoltDB throughput by 15.4 times in comparison to paging to disk. However, nbdX performed same as InifiniSwap on VoltDB.
    b. Facebook workloads on Memcached - memaslap, a load generation and benchmarking tool for Memcached was used to measure performance using recent data published by Facebook. The experiment was done on /etc and /sys partition with different rates of SET operations on Memcached. Initially, 10 million SET operations were used to populate a Memcached server and then in second phase another 10 million GET queries was performed. /etc was GET heavy operations while /sys was SET heavy operations. It was observed that with InfiniSwap, the performance was steady instead of facing linear or super linear drop, infact InfiniSwap improved Memcached throughput by 4.08 times 15.1 times in comparison to paging to disk. Whereas in comparison to nbdX, InfiniSwap improved Memcached throughput by 1.24 times 2.45 times than nbdX.
    c. Twitter graph on PowerGraph - TunkRank algorithm was used on PowerGraph. A Twitter dataset of 11 million vertices was used as an input and a workload of 9GB was created. It was observed that unlike disk, performance using InfiniSwap was stable. With InfiniSwap, PowerGraph only observed 1.24 times higher completion time than 8 times using disk. InfiniSwap improves PowerGraph’s completion by

6.5 times in comparison to paging to disk; on the contrary nbdX did not even complete at 50% of the peak usage.

d. Twitter data on GraphX and Apache Spark - Apache Spark 2.0 was used to run PageRank on same Twitter user graph using GraphX and Spark. InfiniSwap improved performance for GraphX by 2 times than paging to disk at 50% configuration. However, for Spark, all the three methods namely, InfiniSwap, nbdX and disk failed to complete for 50% configuration.

3. InfiniSwap daemon - A HeadRoom of 1GB was set and the experiment was started using a Memcached server and InfiniSwap hosting a large number of remote slabs. It was observed that InfiniSwap daemon would proactively evict remote slabs by monitoring memory usage to make room; when Memcached would stop allocating memory, even InfiniSwap would stop evicting slabs and when Memcached would start resuming allocations again, InfiniSwap would resume slab evictions again. Less than 2% of throughput loss was observed and the median time to evict slab was 363 microseconds and the eviction speed of daemon would keep up with the rate of allocations mostly.
4. The above methods were performed individually on each component, however a cluster wide performance was also measured for individual applications. About 90 containers were created (equal number for each of the above mentioned applications) and placed randomly across 32 machines. The distribution of containers' configurations at 100%, 75% and 50% of peak usage was 50%, 30% and 20% respectively. All the containers were started at the same time and their completion times was measured. It was observed that InfiniSwap increased cluster memory utilization by 1.47 times and significantly decreases memory imbalance.

How do the two components of INFINISWAP work? [Key Question]:

As we saw in the "Key ideas of the solution" section, that the two components namely, InfiniSwap block device divides its entire address space in slabs of fixed size "SlabSize"; and InfiniSwap daemon that runs in user space and is mostly responsible for monitoring and managing memory in each remote machine and in performing slab evictions on behalf of local applications and InfiniSwap block device.

In the InfiniSwap block device, each slab will start in an unmapped state and the InfiniSwap monitors the page activity rates of each of those slabs using EWMA. The block device maintains a bitmap for each slabs to determine whether they are mapped in remote memory. These corresponding bits are initially set to zero and are set only when the page activity rate of a slab crosses a threshold "HotSlab" and these slabs are then mapped to a remote machine's memory. Their corresponding bits in the bitmap are reset to 0 when they have been evicted by daemon as explained below or when a remote machine fails or when the block device itself removes a slab from remote memory as a result of its page activity rate going below a threshold "ColdSlab". There are two queues maintained - RDMA dispatch queue and disk dispatch queue; whenever a slab is mapped to a remote machine's memory, the write request is put into both the queues which will then be processed later and the kernel will reclaim physical memory for that slab. For any unmapped slabs, the write request is only put in the disk dispatch queue. Also, for any page read request, if the corresponding slab is mapped in the block device and its bit in the bitmap is set, then the read request is put into the RDMA dispatch queue, otherwise the slab is read from the disk. Often, to optimise I/O requests, these page read or write requests are handled in batch by VMM. The eviction of slab in block device is initiated by InfiniSwap daemon, the other primary component of InfiniSwap, and is explained in details in the next paragraph.

InfiniSwap daemon on each remote machine monitors memory usage of each of those machines using EWMA method with one second period. The daemon always try to ensure that a certain amount of free memory "HeadRoom" is maintained in the machine always by controlling its own memory allocation. Whenever the amount of free memory goes above the HeadRoom threshold, it allocates slab of size "SlabSize" and marks them as unmapped for block device to map its slab to. Whenever, the free memory goes below HeadRoom, the daemon will first start releasing still

unmapped slabs and if the HeadRoom free capacity is not ensured after this then it starts evicting the mapped slabs in a batch manner. Instead of randomly evicting any mapped slab which increases the probability of evicting a busiest slab, therefore in order to evict E slabs to leave more than HeadRoom free memory, the daemon selects E+E' slabs and communicates with the machines hosting those slabs. After the communication, it evicts the least E active slabs out of E+E' slabs. On selecting these E slabs, the daemon sends EVICT messages to their corresponding block devices and once the block devices has reset the slab's bitmap, it responds with DONE message and then the daemon releases the slab.

Strength :

1. It is scalable as it avoids the need of centralised coordination by utilising decentralised techniques during slab placement in block devices and eviction in daemons.
2. It is fault tolerant because of its decentralised nature. Since it does not have a centralised coordinator, there is no single point of failure and even if a remote machine fails, it relies on the remaining remote machines.
3. It does not call for any hardware or architecture or user applications modifications to support memory disaggregation unlike some of the existing solutions since it's implemented as a loadable kernel module in Linux, it can be easily loaded as a module in any linux architecture and the block device could be mounted on any partition and the daemon is run at user space.
4. The slab eviction or the slab placement or mapping technique or algorithm explained in the above sections ensures that thrashing is minimal and that performance of applications is not reduced drastically

Weakness :

1. InfiniSwap cannot transparently emulate memory disaggregation for CPU- heavy workload applications such as VoltDB, unlike any memory-intensive applications such as Memcached, due to the inherent overheads of context switching involved as a result of paging.
2. The "SlabSize" chosen by the InfiniSwap to divide the address space of block devices into fixed size slabs is quite large to reduce meta-data management overhead. Such a large SlabSize can lower flexibility and decrease space efficiency of remote memory.
3. InifiniSwap has not been designed in terms of being application-aware. Having an application awareness can allow InfiniSwap to infer memory access patterns which would enable them to gain significant performance benefits as seen in the case of VoltDB where it did not perform well.
4. InfiniSwap relies on swapping to provide remote memory access without any modifications to OS. However, due to such swapping, there is an overhead of context switching because of which performance of it is not predictable under different workloads for the same application.
5. InfiniSwap assumes that it has no competition with other applications in the RDMA network. However in reality, network is always a bottleneck because as number of applications using RDMA increases, contentions for the network will increase and will impact the performance of InfiniSwap drastically.

Follow Up Works -

1. Remote regions: a simple abstraction for remote memory [6] - The paper proposes an abstraction in Linux Kernel which provides a simpler interface to RDMA. The paper proposes an idea that a process can export part of memories as files called remote regions that can be accessed through the usual filesystem.
2. LITE Kernel RDMA Support for Datacenter Applications [7] - The paper proposes LITE, a Local Indirection TiEr for RDMA in the Linux kernel that virtualizes native RDMA into a flexible, high-level, abstraction and allows applications to safely share resources. Despite the belief that kernel bypassing is essential to RDMA's low-latency performance,

the paper shows that using a kernel-level indirection can achieve both flexibility and low-latency, scalable performance. The authors developed several popular datacenter applications on LITE, including a graph engine, a MapReduce system, a Distributed Shared Memory system, and a distributed atomic logging system to showcase the benefits of LITE. They demonstrated that their implementation of PowerGraph uses only 20 lines of LITE code, while outperforming PowerGraph by 3.5x to 5.6x.

References -

1. Efficient Memory Disaggregation with Infiniswap - Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang Shin
2. Network Support for Resource Disaggregation in Next-Generation Datacenters - Sangjin Han, Norbert Egi, Aurojit Panda, Sylvia Ratnasamy, Guangyu Shi and Scott Shenker
3. Disaggregated Memory for Expansion and Sharing in Blade Servers - Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, Thomas F. Wenisch
4. A Transparent Remote Paging Model for Virtual Machines - Haogang Chen, Yingwei Luo, Xiaolin Wang, Binbin Zhang, Yifeng Sun
5. Accelerating Relational Databases by Leveraging Remote Memory and RDMA - Feng Li, Sudipto Das, Manoj Syamala, Vivek R. Narasayya
6. Remote regions: a simple abstraction for remote memory: Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguilard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, Michael Wei - https://dl.acm.org/citation.cfm?id=3277430
7. LITE Kernel RDMA Support for Datacenter Applications **-** Shin Yeh-Tsai, Yiying Zhang - https://dl.acm.org/citation.cfm?id=3132762