Review of Paper 2
by- z5284381

DiLOS: Adding Performance to Paging-based Memory Disaggregation

Summary:

Problem being solved:

This research paper critically analyses the concept of memory disaggregation which is the process of separating the computer and the memory onto separate nodes. This process of decoupling enables us to utilize the full potential of the memory in the datacentres. Memory disaggregation potentially solves the problem of memory imbalance which results in enormous and redundant paging as well as thrashing on the cloud servers and data centres. The paper in general tries to put forth that page-fault over head in general os is not actually a fundamental limitation here. The key issue that increases the need for more research and development in this field is the improper utilization and overload causes on memory resources by the data intensive applications. These applications store large amount of data, and the continuous use of the applications limits the space of the available memory at an increasing rate. When the idle memory is no longer available then these applications suffer from page faults and performance loss. This paper provides an in-depth analysis of the kernel-based approach of memory disaggregation and argues that the method is not a compatible method due to the excessive page faults occurring because of memory imbalance. Library based approaches are another way of handling memory imbalance which overcomes the limitation of kernel-based approaches, but the method is not feasible due to its inability to maintain generality. The paper therefore tries to find an efficient solution which aims to find profound solutions to efficient disaggregation of memory and overcome the limitations of the existing methods mentioned above.

Solution addressed:

To solve the issue concerning the issues mentioned above the paper proposes the creation of a paging-based memory disaggregation system DiLOS built on a unikernel. DiLOs, is a new memory disaggregating unikernel(this is quite important here) which intends to deliver performance and generality. DiLOS tries to create a general lightweight page fault handler on top of the unikernels simple execution model parallels with effectively trying to utilize app-aware prefetching for extra optimization as the unikernel serves a single application at a time. The research paper tries to depict that that DiLOS outperforms a recent library-based

system (AIFM) (which looks like have some issues with the numbers associated looking into it with the caching sizes of varying inputs). DiLos just because of its memory disaggregation appears to show better performances than state of-the-art paging-based system but even higher throughput (always look whether this has been seen correctly) of Redis by 27% using the app-prefetcher.

Background and Motivation:

Resource disaggregation has found out to be the key driver in solving the chronic resource under-utilization problem in datacenters. To solve this, Cloud service providers are building disaggregated datacenters (DDC) to take full advantage of resource disaggregation. Memory disaggregation being a subset of the resource disaggregation enables building a large memory pool shared across compute nodes and overcomes the memory wall in the cluster of traditional server node.
Memory disaggregation have been built as kernel based features and parallely library based features have also been used for the addition of the memory based disaggregation. It has been in general seen that kernel based memory disaggregation has been a bit more expensive because of costly switching overheads and parallely library-based approaches trade compatibility for performance. Then we observe unikernels as a plausible solution to the issue of memory disaggregation for the datacenter like environments as they break the boundary between a kernel and a process and integrate them together and have a rower attack surface and provide a robust isolation. Moreover, unikernels open doors for additional improvements by leveraging application semantics; tailoring a kernel to an application includes optimization on paging techniques such as app-aware prefetching following which the author conjecture that it makes DiLOS a perfect choice for the memory disaggregation problem as offers both performance and compatibility and tries to make it possible for it to have a very low cost page fault and also at the same time utilize application semantics without any application modifications.

Design and Implementation:

DiLOS consists of the four key components namely fast page fault handler, prefetcher, page manager, and communication module and it runs on a computing node and a memory server on a memory node. On a computing node, local physical memory works as cache for remote memory similar to many modern disaggregated systems. The kernel here is a Library OS. An application inside the system contains an app-aware prefetching guide. The guide uses the application's domain knowledge to provide semantic hints for prefetching. For the purpose of the Fast page Fault handling and the key idea behind reducing the page fault latency is overlapping the page fault with asynchronous network requests. Local and the remote page tables are combined to a unified page table. DiLOS walks the unified page table and reads the embedded information. If the PTE is a remote address entry, DiLOS fetches its page from the memory node. For protected PTEs, which means there is an outstanding fetch request, the page fault handler waits for its completion. DiLOS cuts down on page faults during prefetching via direct mapper.

Instead of storing prefetched pages into the swap cache, it directly maps them to the page table. This design reduces the number of page faults and saves the swap cache lookup latency. DiLOS has another way to track the statistics: a hit tracker. Upon prefetching, the hit tracker directly reads accessed bits in the page table entries and stores accessed addresses in access history. Then, it uses the information to induce a hit ratio. It uses a mixture of the clock replacement algorithm and LRU list if the local memory has no room to allocate new pages. It uses two background threads, a cleaner and a reclaimer for the page eviction. RDMA is being used through a RDMA driver which is custom written and it has two paths :a control path for managing RDMA resources and a data path for transferring data. The memory node runs as a memory server as a process. It reserves memory using 1GB huge TLB pages and registers them as RDMA memory regions and the control requests received are just translate back to the driver on the ost side which translates the addresses in the requests to map MMIO region. After the mapping, DiLOS issues requests to the RDMA NIC (RNIC) using the MMIO region without the driver's intervention. Huge pages reduce cache misses in the RDMA NIC; thus, they reduce the number of address translations in the NIC. Unikernels can also use the application level semantics to customize the kernel's operations. We know that In unikernels, on the other hand, the costs of system calls and upcalls are the same as function calls so that an application can specify user-level policies requiring frequent communication between the application and the kernel. DiLOS also provides upcall/downcall interfaces for prefetching guide which customize its prefetching algorithm via the interfaces and when the [page fault happens now it invokes a fetch request to communication model and then the guide uses the semantics to inform the prefetcher to fetch the data required. DiLOS also provides the concept of the subpage prefetching when an app-aware guide quickly needs a small amount of data in a memory node consequently accelerating prefetch decisions.

It uses the ddc_malloc which is a drop-in-replacement of default memory allocator and a custom elf loader as well. The ddc_malloc identifies the memory pages that can be evicted to a remove memory server. During loading an application binary, DiLOS ELF loader links malloc in PLT and GOT sections to the ddc_malloc, making application use disaggregated memory. DiLOS is created on top of OSv and has all the above mentioned features but has some limitations as well as it lacks support for the multi - user APIs such as fork(), vfork(), and clone() and no fault tolerance mechanism is present.

Performance/Results/Evaluation:

The system configurations have been very well described which is always a positive sign while benchmarking the code but at some places certain assumptions have been made to match the requirements of the results that are required to be shown that the system is actually better than either a library based system or a kernel based system.

The major problem inside the benchmarking primarily was that the diagrams were a bit unclear as to what they were depicting and the written explanation did not made coherent sense with the

what the diagrams were trying to depict and there was no mention of the individual memory disaggregation solution inside the charts itself.

- **Data analytics application**: Peak memory usage of 40GB and comparison of DiLOS , AIFM and Fastswap is being done

  .DiLOS, as well as Fastswap, is 30% faster than AIFM for 100% cache due to preserving memory abstraction in the kernel.

Verdict and utility:

In this paper, the authors have conducted a comparative analysis between the traditional methods viz. Kernel method, library-based methods and the proposed method of Dilos. The authors claim that the Dilos method is better and more accurate method because it maintains the generality and handle page faults because of its unique design which consists of a fast and lightweight page fault handler at the top of the unikernel. In order to prove their claims the authors applied the Dilos on OSv(17) and compared its performance with the traditional kernel, library based, AIFM, Fastswap methods. The results of the comparative performance analysis reveals that the proposed Dilos method is superior to the performance shown by other methods. Based on the results the paper concludes that Dilos method is the superior method of memory disaggregation and must be used to solve the problems of memory disaggregation as it does not comprise compatibility, it demonstrates generality and avoids the limitation of excessive page faults.

Pros:

- Solves the huge problem of increasing page fault rates inside the system and tackling it using the part of the semantics of a unikernel and part of the application behaviour.
- Overcomes the memory wall in the cluster of traditional server nodes.
- The memory disaggregation has quite a lot of benefits over the traditional data center model which if not able to meet the needs of the user requests leads to significant issues within the memory sub system causing humongous fragmentation by disaggregating the resources especially memory into a separate system on the data centres the processes will be able to meet its needs on the fly and thereby providing more benefits than the usual system and this is the major reason for the memory disaggregation. Memory specific applications will now not only be confined to single servers resources which would have

been eaten up quite quickly before but now its needs can be easily met as the memory is disaggregated into a separate system bringing more agility and modularity in the system.

- The research paper gives relevant background info wherever required for setting up the general tone for the things that are being talked of and solved and references are very well given wherever required.
- Inside the DiLOS System the application code does not require any modification for having the prefetching implemented, as it is implemented as a third party library separately, so this is a generic strength of the system in this case.
- Accepts that the a fault tolerance mechanism when a memory node fails has not been provided.
- Although different configurations underneath have been used for testing but still the acceptance of the divergence of results have been given inside the sections pertaining it.
- The benchmarking tests have been tested for variety of scenarios but unfortunately have only been microbenchmark tests.
- A very well description of the system configuration over which the tests are run ahs been given which is always a positive sign.
- Limitations of the system have been mentioned and the erroneous conditions have been talked about in the paper which is again a positive sign.
- The system configurations and why the certain things like RDMA doesn't work for benchmarking has been very clearly described rathe than keeping the readers under darkness to know what is going around, so great job to the authors there as well.

Cons:

- Not a lot of context about unikernels have been given(and potentially a paragraph explaining the unikernels and the history and the design tradeoffs with the implementations of unikernel could have given a great preface to how DiLOS is being created).
- Only microbenchmark tests have been done, no macro benchmark tests were performed.
- Possibly could have explained what is the issue of ‘memory wall’ which is the major motivating factor for the division of the resources at the disintegrated Data centers.
- Not very clear as to how the library based approaches are being used, what limitations are they solving. Probably a bit more depth of explanation into them could have given more clarity into the motivation behind the creation of the system and what was there method of evaluation.
- No clarity about the ring-0 mode has been given.

Criticism:

- In the section 1 the results have been given which shows that DiLOS performs better than the other alternatives namely AIFM (Library based implementation) and the Fastswap

- There have been some inconsistencies in the explanation of the DiLOS; sometimes it has been said that it in itself is a unikernel. Sometimes it has been said that it is a memory disaggregation system built on a unikernel.

- It is quite unclear in the description of the benchmarks how is the performance being compared and in the images which one of the bars are representing which system(i.e either the images need to be more descriptive or the text explaining the images needs to explain in the image how can we get for the 100% cache utilisation case from the image that DiLOS has a 22- 25% performance drawback).

- No measure of variability (e.g. standard deviation, error bars) is given for any measurement. These benchmarks are likely to be fairly consistent between runs, but some confirmation is necessary.
- Security implications are also not seriously looked into.
- The mention of 184% slowdown for FastSwap is slightly bogus as it does not mention it is slowed down with respect to which state and what cache values.
- Only mentions the results with respect to the relative numbers only.
- It mentions that due to compatibility issues both Fastswap and AIFM runs on different operating systems but there results have been produced as if they have been running on the same system, it slightly rings the bells of deception.
- Different communication modules running for different systems and comparison of the same application will not necessarily give us the complete picture of the performance running application on the system, here there is a lot of disjunction between the underlying configurations being used within the system and that can potentially but not always give us the false hope for a system to work in some conditions.
- It mentions that the performance increases by 1.44x but it is assuming that the time that a system takes in is directly proportional to the performance but actually the results could have been completely different if the machine where the things being tested gets changed, so we cant just talk about the performance in relative terms and with respect to time.

Results Analysis:

As most of the work being done has been clearly stated under which system configurations and which datasets and application it is being the results should be fairly straightforward to reproduce.

Potential Improvements:

First of all the quality of communication and the way the things have been segregated have not been up to the mark, the writeup does not make a lot of logical sense and sometimes things have been just randomly explained without much pre-context. The introduction also does not do much justice in order to explain the things from ground up and spending enough space to explain things. Things like results and the library, kernel based and the exokernel comparisons are done in the introduction but the explanation could have been given more logical pattern. In comparison to the other paper the writeup and the explanation of the generic concepts being used have unfortunately not been upto the mark.