Gmail **Sandeep Kumar <sandeep007734@gmail.com>**

Weekly meeting with Sandeep

13 messages

Prasad, Aravinda <aravinda.prasad@intel.com> Thu, Mar 26, 2020 at 9:40 AM
To: Sandeep Kumar <sandeep007734@gmail.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Moving by 30 mins due to a conflicting meeting.

Rescheduling the meeting to 26th March as 25th is Intel Bangalore holiday.

Dial the toll-free number: 18002092663
Enter Bridge number: 5
Conference ID: 485906974#

....

Join Skype Meeting

Trouble Joining? Try Skype Web App

Join by phone

+1(916)356-2663 (or your local bridge access #) Choose bridge 5.,,485906974# (Global) English (United States)

Find a local number

Conference ID: 485906974
Forgot your dial-in PIN? |Help

....

invite.ics
3K

Sandeep Kumar <sandeep007734@gmail.com> Thu, Mar 26, 2020 at 12:28 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes

We discussed the papers:
Paper 1: **Hierarchical Hybrid Memory Management in OS for Tiered Memory Systems (Memos).**
Paper 2: **Efficient lock-free durable sets**

The assumption made by the Memos paper is similar to what the actual hardware offers. We need to read the paper in more detail, so as to figure out the minute similarities and differences, and whether the current results present in the paper will hold for the real hardware.

The second paper approaches the problem from the other spectrum. Instead of modifying the OS components, it modifies the data structures used to store the data on the memory. The paper presents two solutions, one naive, and one in which the number of flushes required is theoretically lower bounded. The paper also aims to provide a crash recovery, which is not required for our approach. This can be exploited to provide a more optimized version of the data structure.

Another paper on similar lines: https://www.usenix.org/system/files/conference/atc18/atc18-david.pdf

Benchmarks:
We need to find some graph-based applications that can be used to measure the performance, apart from the currently used benchmarks.

[Quoted text hidden]
--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Wed, Apr 1, 2020 at 12:50 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes for 1 April Meeting

We discussed some graph benchmarks.
Bench500 seems to be the most suitable one as of now.

Apart from these, there are a couple of more benchmarks. We need to ensure that these benchmarks do not explicitly depend on GPU or use GPUs extensively, as they can have an impact on the results.

We will explore the Bench500 and benchmarks mentioned in the "Trillion-edges" paper in more detail.

Apart from that, we also discussed the Memos paper. The paper has done a thorough job of designing a memory management module for the DRAM-NVM kind of setting. However, the experiments were done using simulation and hence, they might suffer from some of the caveats mentioned in the FAST paper that has carried out extensive experiments using a real Optane PMM.

We will explore how we can use these observations (by the FAS paper) to improve the Memos working.

Apart from these a couple of ideas:
1. We can design super-optimized memory management of some benchmarks who repeat the same algorithms multiple times. This super-optimized memory management will provide an optimal setting for this benchmark, in a particular setting, but may fail(with high probability) for other kinds of benchmarks.

2. We can also look at durable sets for Optane persistent mode. In this, we design an efficient data structure that reduces the number of flushes required to the main memory.

[Quoted text hidden]
--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Mon, Apr 13, 2020 at 11:55 AM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes on 13th April

We discussed the assumptions used in the Memos paper and compared it with the empirical results.

- There are two main aspects of the Memos: Monitoring of the page utilization, and movement of data.
- During monitoring, the Memos paper uses the memory allocation method of the Linux kernel to observe where the data is being placed and how it is moved across the memory banks.
- They claim that the current allocation is in-efficient, as it does not take it to account cache level collision and has a high bank imbalance (hot data going into same memory bank rows). They have modified the allocator and show that it reduces the bank imbalance and improves the performance.
- We need to study the allocation strategy and see how *general it is.*
- During data swapping, they start with moving 10,000 pages. This might cause a performance dip as it may cause contention at the iMC queue level, and maybe also at the XPBuffer.

In the next meeting, we will discuss in more detail the memory allocation strategy and DMA level details for the Optane. Apart from this, we will also discuss graph benchmarks details.

[Quoted text hidden]

--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Wed, Apr 15, 2020 at 1:41 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting Minutes 15th April

1. We discussed different graph-related benchmarks, and we have shortlisted three benchmarks as of now for our experiments
    1. Graph500
    2. RPAT graph generator
    3. Hyperlink-14
2. The size of these benchmarks is/or can be configured to be more than 384 GB, the current DRAM capacity of the machine, so as some of the data will be spilled in the NVM. This gives us an opportunity to optimize the memory allocation policies
3. We have concluded our discussion on the Memos paper.
    1. The MEMOS paper has used a simulator, which allows them access to the placement algorithms at the bank level, which is typically not available in a real-world system.
    2. Due to this reason, their memory allocator and the cache management policies cannot be implemented on a real-hardware, as of now.
    3. Their HyMM: Hybrid Memory Monitor, on the other hand, does not make any such assumptions and can be used to determine the hotness and coldness of a page, along with its read/write properties.

For the next meeting, we aim to explore the HyMM in more detail and see if this can be executed on real hardware. There is an another paper that allows monitoring of TLB misses, "A tool to instrument x86-64 TLB misses", which will be discussed in the next meeting.

We also need to check if a DMA transfer can happen between two DRAMs, and if yes what are its requirements.

[Quoted text hidden]

--

Regards
Sandeep Kumar

Prasad, Aravinda <aravinda.prasad@intel.com> Mon, Apr 20, 2020 at 4:57 PM
To: Sandeep Kumar <sandeep007734@gmail.com>

Hi Sandeep,

Link to an article on DMA and persistent memory:

https://lwn.net/Articles/672457/

Regards,

Aravinda

[Quoted text hidden]

Sandeep Kumar <sandeep007734@gmail.com> Wed, Apr 22, 2020 at 12:07 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes 22nd April 2020

Discussion:

1. We discussed the working of BadgerTrap.
    1. The supported Kernel is 3.12.13, which is giving some error when tried to install on a VM. We will try to install on Qemu using BuildRoot and see if that works.
    2. The page-fault technique that they have is used is in-efficient. The paper claims an overhead of 3X to 40X. We can use it to profile the applications, but we cannot use this on a production system.
    3. The paper also lists some limitations regarding capturing ITLB misses, and the kernel-level misses. We will explore if there is a way to do this.
    4. We will also explore how to port it to a recent version of the kernel.
2. Perf tool can be used for sampling the TLB misses, and if the debug symbols are present, then we can get the addresses also. However, it produces a huge amount of data and is not practical for long-running applications.
3. A DMA transfer from DRAM to an NVM, in the DRAM slot, is not possible. It requires someone to generate the addresses during the copy, which in the absence of a DMA controller, is done by the CPU.
    1. However, we can optimize the copy of the pages across memories by using techniques from the Nimble page management paper.
4. A binary classification for the hotness of a page may not work. We need to classify the pages in a more granular manner, like a *hot page index,* that spans from 0 to 100. This will provide finer control of the movement of the pages.
5. For a huge page, it might be possible that only a part of the page is hot, and not the complete page. Hence, it might be worth splitting the huge page and send cold 4K pages to the NVM and keep the hot ones in the DRAM. Need some experimentations to actually quantify the effect of doing so.

Next meeting:

1. Explore the badger trap tool, see if it runs on Qemu.
2. Nimble page management paper discussion.
3. Need more detail on the hotness index.
4. How can we use the properties of the Intel Optane to actually make better decisions while moving the data?

[Quoted text hidden]
--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Fri, Apr 24, 2020 at 12:58 AM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>

Hey Aravinda,

I am in the process of trying to boot the kernel with support for Badgertool.

As per the documentation, BadgerTool supports Linux kernel 3.12.13. Building this kernel requires GCC-4.8 as some of the latest changes in GCC fails the build process.

1. Using Qemu: The building process (after patching the kernel code) works fine. However, Qemu fails to boot. A similar step with the latest Kernel 5.3 works just fine.
2. Using BuildRoot. The latest BuildRoot does not give have GCC-4.8 support in its toolchain. Because of this, the kernel build fails.
3. At this point, I think it will easier to port the BadgerTrap code to the latest Kernel. I took a glance at the code, and it looks feasible to me

Please let me know what you think. What will be the best way to proceed with this.

[Quoted text hidden]

--

Regards
Sandeep Kumar

Prasad, Aravinda <aravinda.prasad@intel.com> Fri, Apr 24, 2020 at 10:04 AM
To: Sandeep Kumar <sandeep007734@gmail.com>

If porting is easy, I prefer porting to the latest kernel. This will help as we will be evaluating our benchmarks on the latest kernel.

From: Sandeep Kumar <sandeep007734@gmail.com>
Sent: Friday, April 24, 2020 12:59 AM
To: Prasad, Aravinda <aravinda.prasad@intel.com>
Subject: Re: Weekly meeting with Sandeep

Hey Aravinda,

[Quoted text hidden]
[Quoted text hidden]

Sandeep Kumar <sandeep007734@gmail.com> Thu, Apr 30, 2020 at 8:13 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes 29th April

Discussion:

1. We discussed the page migration techniques from the Nimble paper.
2. The paper has argued to use THP (transparent huge pages) and threads to increase the performance of the page movement. Specifically, the authors suggest 4 optimizations:
    1. Use THP
    2. Use Threads
    3. Use concurrent page migrations
    4. Use the swapping of pages.
3. We discussed the effects it might have on the cache. Migrating large amounts of data will definitely pollute the cache. It may affect the performance of the benchmarks. Hence, we need to quantify this effect. A simple way will be using BadgerTap and check the TLB misses from the application, by varying the number of migrating threads.
4. The Fast paper has suggested using non-temporal stores such as *"ntstore"* instructions to move a page to the NVM memory. Doing so will not pollute the caches, however, they may also not get the benefit of several hardware optimizations of moving a page.

    1. We need to find a good balance between temporal and non-temporal stores.
5. Furthermore, in a VM kind of setting, where many applications are contending for the CPU cycles, using a large number of threads to migrate the page may affect the performance of those applications. Thus, nullifying the whole aim of migrating the pages -- to improve the performance of the applications.
6. We need to be careful when to use threads, based on the current CPU, memory, and IO usage of the system.
7. We can have different policies for moving a page from fast to slow memory and slow to fast memory, as they will be carried out in different scenarios.
8. We are in the process of porting the BadgerTrap code from kernel 3.12.X to 5.3X
9. The source code for Nimble page management is available. It is current on 4.X kernel, with an RC branch on GitHub for 5.X kernel.

Next meeting

1. We will discuss the paper "Bridging the Latency Gap between NVM and DRAM for Latency-bound Operations".
2. How to use the "hotness index".
3. Code progress for porting the BadgerTrap tool.

[Quoted text hidden]
--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Thu, May 7, 2020 at 8:16 AM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes 6th May:

Discussions

1. In the last meeting, we discussed the coroutines paper. The authors leverage the "suspend and resume" capabilities of a coroutine to hide the latency induced by using NVM.
2. The authors show that for a particular class of applications, databases in this case, that are computationally intensive, one can hide the latency induced by accessing an NVM memory by scheduling other work till the IO operation returns.
3. The work done in the paper is orthogonal to what we have discussed so far. It does not make any kernel-level changes and aims to improve the latency of NVM using applications purely in the userspace.

Directions
Based on our discussions, we have identified a few research directions

1. Hotness index: Track the hotness index of a huge page, and see if it is a "densely" packed hot huge page? If the whole huge page is not hot, we can split the huge page and move the cold parts to the NVM memory and keep the hot part in the DRAM. We may want to combine hot pages and promote it to a huge page.
2. Measuring the hotness of a base page is tricky. As pointed out by Aravinda, even if we are incrementing a counter in the page structure, the whole page will be marked as hot. However, only a small part of the page is frequently accessed.
    a. Because of this reason, the page will be in the caches, and we will rarely see a TLB miss for this.
    b. While executing an application, the kernel will only come into the picture when there is a page-fault, i.e., the page currently being accessed is not in the memory or is not valid.
    c. We can force a TLB miss by manipulating the page-table, which is under the OS control. Many profilers - and some malicious applications- use this technique. However, it has a severe impact on the performance of the applications (up to 40X as per BadgerTrap).
    d. Also, manipulating a base page for a C like code is tricky, as it may cause SEGSEV due to the excessive use of the pointers.
    e. A java like application, where the JVM oversees all the memory accesses, might be easier to do.
3. We can explore the hardware counters to get stats for a based page.

Next Meeting

1. We will discuss an efficient way of tracking a 4KB page and how to make modifications to it without affecting the application.
2. We are still in the process of porting the BadgerTrap code to a 5.X kernel.

[Quoted text hidden]
--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Tue, May 12, 2020 at 6:48 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>

Hi Aravinda,

I was speaking to Ashish about his paper Hawkeye while exploring methods to efficiently track usage of huge pages. He suggested a paper for memory pointer modification in C/C++ : Mesh: Compacting Memory Management for C/C++ Applications

Shall we discuss Hawkeye, Ingens, and Mesh from the point of view of efficiently tracking a base page tomorrow?

Sorry about such a late mail. We can discuss a broad outline tomorrow and maybe go into details more in subsequent meetings as per the requirements.
[Quoted text hidden]
--

Regards
Sandeep Kumar

Sandeep Kumar <sandeep007734@gmail.com> Wed, May 13, 2020 at 9:26 PM
To: "Prasad, Aravinda" <aravinda.prasad@intel.com>
Cc: "Subramoney, Sreenivas" <sreenivas.subramoney@intel.com>

Meeting minutes for 13th May 2020:

Discussion:

1. We briefly discussed the Mesh paper. The main idea of the paper is to reduce the memory bloat in an application. It does so by tracking the object allocation offsets within a page. If two pages are disjoint in their offsets used, they are candidates for *meshing.* Mainly, a meshing of two pages involves copying of data from one page to another, and free the first page.
2. We discussed the idea of *book-keeping* pages. Essentially, these pages are accessed frequently. However, only a small amount of data is accessed from it. This is an issue, as a page is hogging down the expensive DRAM memory when this can be stored in the NVM memory.
3. Moving the page to the NVM solves the bloating issue. However, it increases the latency of those small accesses. It might incur huge performance overhead if those small accesses were done more frequently.
4. To solve this issue, we can use a small structure in the DRAM that will track is small updates and will flush it to the NVM asynchronously.
5. This idea needs to be verified using some characterization experiments. Furthermore, as suggested by Aravinda, even if the application does not have a significant amount of book-keeping page, the kernel can re-allocate or re-arrange the data so that it has some book-keeping pages. We can use it to optimize data placement, by aggregating all the small accessed data in a single page (stored in DRAM) and the rest of the data, which are not accessed that frequently, flushed to the NVM.
6. Badgertrap can be modified to profile the book-keeping pages. I have sent a mail to the authors, asking whether they have an updated version available. Update: I got a reply from Professor Arka; they have not updated the tool for newer kernel versions. I will send a mail to Wisconsin-Madison and see if they have done any work on this.

Next Meeting

1. As of now, we have enough directions on which we can start working. We will properly list out those directions, and pick one for a more in-depth study, like what are the challenges in this case, how practical it will be for different scenarios.
2. We need to profile an application for various memory-related statistics. BadgerTrap is a good tool. We can use this as a base and modify it as per our requirements. We need to check the license and make sure Intel is ok with using this.

[Quoted text hidden]

--

Regards
Sandeep Kumar