Rescuing the Future of Memory with Data-Oblivious Algorithms

Sitota E Mersha
Addis Ababa University

Nicholas Wendt
University of Michigan

Valeria Bertacco
University of Michigan

Abstract

Data generation is accelerating at a very fast pace worldwide. A large fraction of generated data is processed in data centers, being analyzed to extract relevant information from it. Yet, our ability to process large data and store it remains limited. Research suggests the use of Disaggregated Data Center (DDC) architecture [1] which efficiently utilizes computing resources through segregation on different racks. These resources are connected through a network and allocated to each application based on demand, which increases efficiency. However, a key challenge in leveraging these pools efficiently is the severe overhead caused by the high latency of moving data to and from compute nodes over the internet. While in a traditional server, data is stored right next to the compute node making the latency relatively small. This work explores the application of Data-Oblivious Algorithms in a Disaggregated Server environment. Data-Oblivious Algorithms [2] [3] are algorithms whose behavior does not depend on input data and have a deterministic computation sequence which can be used to predict the next memory access with the goal of minimizing the overheads and enhancing performance. We focus on replacing the Least Recently Used Algorithm with an implementation we call Oracle, that can generate an optimal schedule for paging. Additionally, we implemented fetching data from a remote node ahead of time extending Oracle, to a scheme we call Oracle + Prefetch. We evaluated the speedup of these schemes alongside the Least Recently Used implementation and resulted in Oracle having 1.7x speedup and Oracle + Prefetch had a 2.5x speedup.

1 Overview

According to the Disaggregated Data Center vision [4], computing units (CPUs, GPUs, accelerators) are segregated from the memory hierarchy and connected by the data center network fabric [1] instead of a system bus. Examples of these systems include the Facebook Disaggregated Rack [5], HP "The Machine" [6] and SeaMicro [7].

By decoupling resources, DDC makes it easier for each technology to evolve independently [8] reducing the time-to-adoption, redoing integration and motherboard design. In addition, disaggregation enables fine-grained and efficient provisioning and scheduling of individual resources across jobs. Disaggregation focuses on just-in-time resource allocation to stop the rise in infrastructure expenses. When a workload calls for resources, easily accessible compute or memory components are removed from where they are underutilized and added to where they are needed the most on the fly, without deploying additional servers or adding excessive capacity. Contrary to traditional (i.e., aggregated) data centers, where adding compute capacity necessitates the addition of entirely new servers with expensive memory and networks, which lead to underutilization of resources and increasing costs for additional space, power, and management overhead. The concept of server disaggregation is catching on in data centers as it delivers significant benefits, including faster and less-expensive server upgrades, pooled server resources, and banks of servers specifically designed for almost any workload.

However, a key challenge in implementing disaggregated servers efficiently is the severe overhead caused by the high latency of moving data to and from compute nodes over long network fabric (short-distance, high-speed interconnections over high-grade cables such as fiber) compared to a traditional server where all components are on a computer's motherboard incorporating a central processor and main memory and moving data to and from compute nodes uses the underlying circuit (bus) for communication.

We can overcome this bottleneck by exploring new ways to increase the speed of memory accesses and improve overall performance. This can be achieved through efficient memory management techniques as many computing workloads are memory-bound. [9]

One widely used approach in memory management is page replacement. Since local memory cannot hold all the referred pages in memory, page replacement algorithms are needed to decide which page needs to be replaced when a new page comes in from remote memory. Whenever a new page is referred and not present in lo-

cal memory, page fault occurs and the underlying system replaces one of the existing pages with a newly needed page. The goal of any page replacement mechanism is to minimize the number of page faults.

Least Recently Used (LRU) algorithm is a page replacement technique used for memory management. According to this method, which is based on locality of reference, pages are organized in order of use and the page which hasn't been used for the longest amount of time is replaced.

The theoretically optimal page replacement algorithm (also known as Bélády's optimal page replacement policy [10]) is an algorithm where the operating system replaces the page whose next use will occur the farthest in the future. This algorithm could not be implemented in a general purpose operating system because it is impossible to compute reliably how long it will be before a page is going to be used.

We can, however, obtain the list of future memory accesses using the deterministic property of Data-Oblivious Algorithms. An algorithm is called data-oblivious [2] [3] if its control flow and memory access pattern do not depend on its input data. Data-Obliviousness has great applications in cryptography and security. Computation on encrypted data must be Data-Oblivious, meaning that the attacker cannot learn anything about the data using standard side channels such as timing, cache performance, and termination behavior. The deterministic property of oblivious algorithms will always execute the same sequence of instructions, taking a consistent amount of time and performing memory accesses at the same indices no matter input data. In this project we implement Bélády's Optimal Algorithm using Data-Oblivious predictability to come up with an optimal schedule to enhance performance through improved memory management.

We also aim to improve application performance in disaggregated servers where memory latency is high by using prefetching to achieve similar or higher performance in the data-oblivious form [11]. To this end, computing systems use prefetching to bring data close to the processor before it is required. The effectiveness of prefetching heavily depends on how well an application's memory accesses can be predicted as mispredictions would waste memory system bandwidth and capacity.

2 Design and Implementation

2.1 Environment

We consider simulating a disaggregated server infrastructure where memory, storage, and compute nodes are segregated on different racks and connected over a network. For efficient execution of a workload, data should be fetched from remote memory and stored in local memory, so that it can be accessed at run time to avoid latencies.

We implemented a paging scheme called Oracle which is based on Bélády's optimal algorithm and the knowledge of future memory accesses due to the deterministic nature of Data-Oblivious algorithms.

We also implemented an extended version of Oracle called Oracle + Prefetch, which runs parallel from the executing program to prefetch (loading ahead of time) pages from remote storage ahead of time, eventually enhancing performance.

The factor of fetching ahead is a sensitive parameter as over fetching can hurt performance due to a large number of page faults.

2.2 Assumptions

Assumptions for latency values used during this implementation include 15 nanoseconds for local page access latency (read or write), 10 microseconds for remote page fetch latency 10 microseconds for dirty page eviction latency and 0 seconds for clean page eviction latency. With these assumptions made, we calculate the total time to run the program, total time spent on local accesses, total time for remote accesses, total time for dirty eviction, number of replacements, and in the case of prefetching, in addition to the previous values, we calculate the total stall (time the program is idle waiting for fetching from a remote server to be done).

2.3 Running the Simulator

Executable versions of Data-Oblivious Algorithms are run through Valgrind's lackey example tool to obtain memory traces. The trace is fed into our simulator that maps pages based on an input page size provided by the user, and results in a profile that can be run in 3 different modes of execution. These are:

1. lru - implemented based on the least recently used algorithm, simulates paging replacing the page which hasn't been used for the longest amount of time. This mode does not utilize the knowledge of future memory accesses.

2. orcl - our implementation of Oracle an optimal paging schedule utilizing the knowledge future page accesses.

3. orclpref - Oracle + Prefetech , extended version of oracle that fetches pages from remote memory ahead of time.

3 Evaluation

To evaluate the performance of implementations, we used a machine with a 3.8GHz, 8-core x86_64 Intel processor. We tested local capacity as a percentage of total unique pages accessed by the program with page sizes of 4KB, 8KB and 16KB respectively.

We evaluate the performance of Oracle and Oracle+Prefetch against the Least Recently Used implementation using six benchmarks from VIP-Bench [12], a recent research work that models a collection of privacy focused benchmarks with different workloads and operational complexity, implemented using native C++ types while each algorithm was modified to adhere to data-oblivious restrictions. The algorithms used include:

- Bubble Sort - Sorts an array of integers using the bubble sort algorithm.
- Distinctness - Tests an array of integers to see if they are all distinct, returns the first duplicate.
- Flood Fill O(N)2 - Performs a graphical flood fill operation on a 2D privatized image, but this version is more clever and achieves O(N)2 data-oblivious performance.
- Image Filter - Applies a sepia-tone filter to an image.
- Knapsack - Computes the 0-1 Knapsack solution for a set of weights and values.
- NMIST CNN - A 7-layer CNN model performing training and inference on the MNIST handwriting data set.

3.1 Optimal Factor for Prefetching

In this particular implementation, a factor of 750% of fetching ahead was used, since it performed in an average range compared to other percentages computed with varying local capacity as shown in Figure 1.

3.2 Speedup Analysis

The metric plotted is "Memory Access Time Speedup vs. LRU". This takes Least Recently Used as the baseline, since it can be implemented without knowledge of future accesses, and computes speedup (speedup = (baseline time) / (other time)) for Oracle and Oracle + Prefetch. For example, if Oracle completes in half the time of LRU, it has a 2x speedup (2 = 2 / 1). The geometric mean is used to compute the average of multiple speedups.

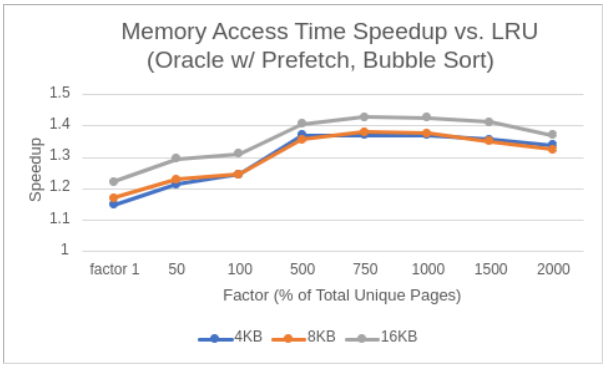

Figure 1: Performance Analysis of Bubble Sort with 1000 random elements. Factor of fetching ahead altered starting from 1 to a percentage of total unique pages accessed by the program.

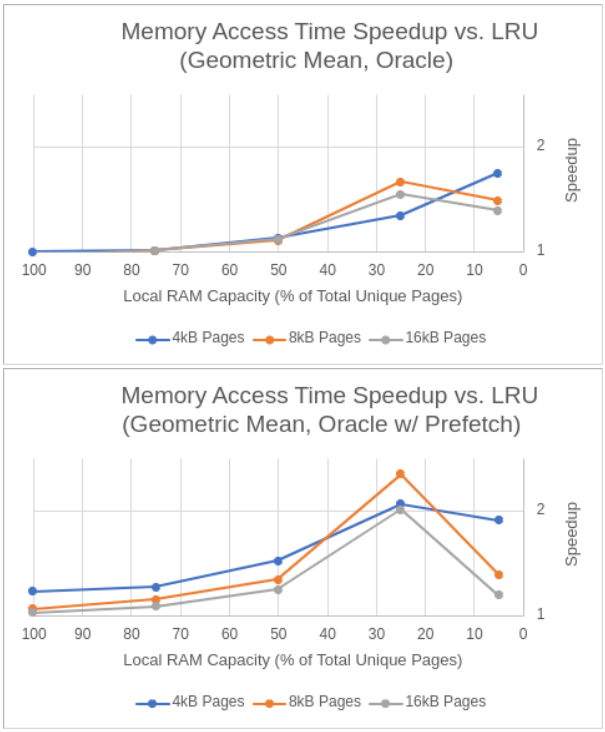

Figure 2: Performance Analysis of geometric means of all 6 algorithms tested on page sizes of 4KB, 8KB and 16KB page sizes compared over varying amount of local capacity as a percentage of total unique pages accessed by the program in Oracle and Oracle + Prefetch.

We wanted to show behavior of algorithms with different input data sets, but since all algorithms are Data-Oblivious we simulated this effect by examining performance over different %-of-unique-pages ram capacities. As the percentage of local RAM Capacity decreases, total time to run the program increases,since more and

more pages have to be swapped to and from remote memory.

Comparing the geometric mean of all 6 algorithms to see how page size affects speedup, shown in figure 2, 8KB page size seems to attain peak performance at a local RAM Capacity value of about 25 % of the total unique pages accessed by the program in both the Oracle and Oracle + Prefetch speedups compared to a base line of the Least Recently Used Implementation.

We have noticed that there are a few classes of algorithms with similar code structure(in this case Bubble sort and Distinctness, Flood-fill and Knapsack and Image Filters and MNIST-CNN), which show identical trends when tested over 4KB pages. The results for speedups of Oracle and Oracle + Prefetch are recorded compared to the Least Recently Used Algorithm as a baseline and shown in figure 3.

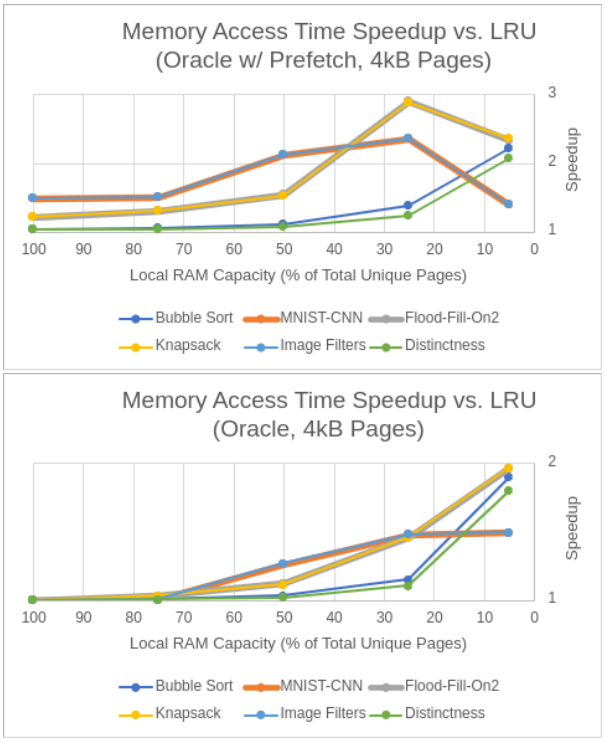

Figure 3: Performance Analysis of 6 algorithms tested with page size of 8KB over varying amount of local capacity as a percentage of total unique pages accessed by the program.

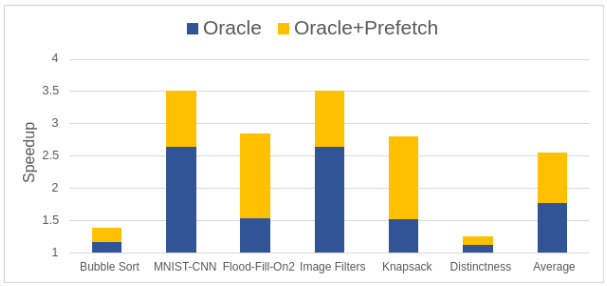

Figure 4: Average Memory Access Speedup by Oracle of 1.7x and Oracle + Prefetch 2.5x compared to Least Recently Used baseline.

4 Limitation and Future Works

We utilized a constant value to represent the remote memory access latency, but we intend to explore how it changes when the same page is accessed repeatedly. We are developing a system that will alert the underlying system when to precisely fetch pages for optimal efficiency. As 750 percent of the total unique pages accessed by the program was used in this experiment due to its average performance, we are continuing research on what the precise factor is for prefetching in our current implementation. Additionally, we are exploring other prefetching configurations to enhance performance in a disaggregated server setting using Data-Oblivious Algorithms.

5 Conclusion

By effectively utilizing computer resources through segregation on various racks, disaggregated servers can address the problem of data flooding. We can use the predictable nature of Data Oblivious Algorithms to overcome the significant overhead created by the high latency of transporting data to and from compute nodes over a network fabric, and generate an ideal paging schedule that can be used at runtime. We can also further enhance performance using the extended version with effective prefetching.

A Artifact Appendix

A.1 Abstract

We provide the public repository for our implementation, available on GitHub. This artifact includes the Orahecle and Oracle + Prefetch implementations and data oblivious versions of t benchmarks we used for our evaluation.

A.2 Artifact checklist

- Public Repository Link: `https://github.com/sitota49/paging-simulator`
- Evaluation Results: `https://1drv.ms/x/s!An00eBSlcYGbgRcM8ZHSQu_7x34N?e=4jjMoX`

A.3 Description

All information is available at our GitHub repository. We have written a README specifically for running the Artifact, which can be found here: `https://github.com/sitota49/paging-simulator`.

References

[1] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 249–264, 2016.

[2] John C Mitchell and Joe Zimmerman. Data-oblivious data structures. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[3] Jiyong Yu, Lucas Hsiung, Mohamad El Hajj, and Christopher W Fletcher. Data oblivious isa extensions for side channel-resistant and high performance computing. *Cryptology ePrint Archive*, 2018.

[4] Sergey Blagodurov. The time is ripe for disaggregated systems, Sep 2021.

[5] Jason Taylor. Facebook's data center infrastructure: Open compute, disaggregated rack, and beyond. In *Optical Fiber Communication Conference*, pages W1D–5. Optical Society of America, 2015.

[6] Kimberly Keeton. The machine: An architecture for memory-centric computing. In *Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*, volume 10, 2015.

[7] Ashutosh Dhodapkar, Gary Lauterbach, Sean Lie, Dhiraj Mallick, Jim Bauman, Sundar Kanthadai, Toru Kuzuhara, Gene Shen, Min Xu, and Chris Zhang. Seamicro sm10000-64 server: building datacenter servers using cell phone chips. In *2011 IEEE Hot Chips 23 Symposium (HCS)*, pages 1–18. IEEE, 2011.

[8] Sangjin Han, Norbert Egi, Aurojit Panda, Sylvia Ratnasamy, Guangyu Shi, and Scott Shenker. Network support for resource disaggregation in next-generation datacenters. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pages 1–7, 2013.

[9] Irina Calciu, M Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 79–92, 2021.

[10] Belady's optimal replacement algorithm, Jul 2022.

[11] Hasan Al Maruf and Mosharaf Chowdhury. Effectively prefetching remote memory with leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 843–857, 2020.

[12] Lauren Biernacki, Meron Zerihun Demissie, Kidus Birkayehu Workneh, Galane Basha Namomsa, Plato Gebremedhin, Fitsum Assamnew Andargie, Brandon Reagen, and Todd Austin. Vip-bench: A benchmark suite for evaluating privacy-enhanced computation frameworks. In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 139–149. IEEE, 2021.