A Study of NPB and CANDLE on Commercial Off-the-Shelf Disaggregated Memory

Melanie Cornelius, Avery Peck, Zhiling Lan
Department of Computer Science
Illinois Institute of Technology
Chicago IL, USA
melanie.e.cornelius@gmail.com. apeck2@hawk.iit.edu, lan@iit.edu

William Allcock, Brian Toonen
Advanced Integration Group
Argonne National Laboratory
Lemont IL, USA
allcock@anl.gov, toonen@alcf.anl.gov

***Abstract*—Disaggregated memory systems offer enormous computational potential. Understanding the state-of-the-art disaggregated memory architecture for deep learning and scientific applications is especially important; memory is an expensive resource and a bottleneck in scaled deep learning and scientific computing. Our study aims to quickly and reliably answer performance cost concerns about an emergent technology: disaggregated memory. We first propose classifying an application into one of three categories using a simple, system-agnostic, easily-replicated measurement. We then use an application's category to predict performance slowdown when using disaggregated memory. Finally, we verify our model using results obtained on a production, commercial off-the-shelf disaggregated memory system available at the Argonne National Laboratory. This paper will step through the simple measurement technique, present the model for categorization, and then verify the model's effectiveness. Our study offers a simple, reliable mechanism to answer the question - is this application an appropriate fit for disaggregated memory? - reducing the cost uncertainty around this emergent technology.**

***Index Terms*—disaggregated memory, performance analysis, deep learning, scientific computing**

I. Introduction

Both historical and recent efforts suggest a shift toward increasingly disaggregated architecture [1]. While storage has been disaggregated for decades, and computing is increasingly distributed, memory is still frequently accepted as a static (albeit hierarchical) resource. The concept of disaggregated memory is to have a discrete pool of memory connected through a high-speed interconnect and available when and where it's needed. Disaggregated memory systems offer many potentials for advantage, including flexible incorporation of the latest memory technologies, an increasingly hybridized memory hierarchy, and access to on-demand quantities of memory.

This sounds excellent at the superficial level. However, it's well-understood all changes in technology come at a cost. In order to be adopted at a wide scale, it's necessary to have assurance that the cost (both monetary and in developer/researcher time) of using an emergent technology will be outpaced by its benefits. For disaggregated memory, this is a particularly challenging question to answer.

Disaggregated memory systems are generally either built on proprietary hardware (such as with HP's The Machine [7] or IBM's dredbox [8]), or the system exploits the memory hierarchy, using some variety of fast storage instead of DRAM [9]. Disaggregated memory systems are thus a) enormously different in implementation, b) often expensive, b) of limited public availability [10]. There is thus little prior work on a simple, effective model to predict an application's performance when using disaggregated memory. This study is intended to address this problem.

Toward our goal, our study consists of three parts. First, we develop a categorization model that uses a simple, repeatable measurement to classify applications into categories. We then use an application's category to estimate its performance slowdown when running on a disaggregated memory system. We verify our model using using nine benchmarks from deep learning and scientific computing. We categorize each, predict its magnitude of slowdown, and then run the application on a production disaggregated memory system at the Argonne National Laboratory (ANL) [5] [11]. Using our classification model, a user can simply, quickly, and effectively answer the question - *how will my application perform on a disaggregated memory architecture?*

This paper is organized as follows. We start by introducing background and related work in Section II. Section III presents our proposed model for classification. In Section IV, we present experimental results of nine deep learning and scientific computing benchmarks on the production disaggregated memory testbed to verify the effectiveness of our model. Finally, Section V concludes the study.

II. Related Work and Background

A. Prior Work in Disaggregated Memory

There are many studies examining various aspects of disaggregated memory computing. Previous works focus on proposing architectures for disaggregated memory systems [12], creating prototypes with proprietary hardware, and simulation. The initial concept of disaggregated resources, and in particular disaggregated memory, was explored in the '90's. Andersen and Neefe (1994) [13], for example, examined the field's current assumptions and proposed new directions. Lim et al. [3] used memory usage patterns for several datacenters as well as simulation to show that disaggregated systems can provide substantial performance-per-dollar improvements where memory is a constraint. Rao & Porter [14] found that disaggregated memory was an achievable, efficient architecture for Spark SQL analytic queries; they used memory access data from a Spark SQL instance to simulate how a disaggregated setup may have negligible performance decrease. And Quiroga et al. [15] studied a disaggregated memory prototype to find potential bottlenecks in using disaggregated memory. They used a custom framework based on IBM's dReDBoX architecture [8] consisting of memory and compute nodes orchestrated by a controller. Because the compute-only nodes retain no local memory, this system can be regarded as a fully-disaggregated memory system. Their results highlighted that disaggregated systems would need to carefully consider network bottlenecks on data requests; they proposed packetization protocols could contribute important performance improvement.

For deep learning in particular, Kwon & Rhu [16] propose a deep learning-optimized disaggregated memory system that focuses on providing GPU/NPU access to remote shared memory. Their simulations make a case for the scalability of a system dedicated to providing remote memory to accelerator devices.

Distinguishing from these existing studies, our work is one of the first to study application performance on a production disaggregated memory testbed.

B. RAN

The RAN project at ANL is an initiative to try reshaping how RAM is used [17]. The RAN project is currently attached to Cooley, a visualization cluster at ANL [18]. Here, each of the 126 compute nodes has an NVIDIA Tesla K80, two 6-core, 2.4-GHz Intel E5–2620v3s, one Mellanox InfiniBand FDR card, and 384 GB RAM per node (see Fig. 1). The three remote memory devices, together referred to as the "remote pool", have a cumulative 6 TB RAM, and 4 or 6 FDR InfiniBand cards each (see Fig. 2).

This memory is a scheduled resource capable of assignment to any number of nodes at a time. When memory is assigned to a node or set of nodes, we call this "allocation". The memory is removed from the remote pool, and the node or nodes use their local memory as a cache for access into the remote allocation. The system is a write-back cache, where the local RAM cache on a node is backed up to the remote allocation when certain events trigger. All memory from the local cache is eventually backed up to the remote device, though at any time, the local cache may only have some subset of the node or process's memory present at a time.

In essence, the RAN project treats local RAM as an extension to the memory hierarchy, using some or all of a node's local RAM as a cache to the remote memory allocation. This resembles some prior work [9] but differs in that the remote devices are both COTS and consist of actual DRAM allocated across the InfiniBand network.

III. Classification Model

In this section, we present our simple classification model and discuss how to use it. Our model is built on an application's major page fault behavior over time. We first discuss the mechanism of measuring major page faults and the ideas behind this choice. We then discuss our categorization model that classifies applications into three categories according to their major page fault patterns. For each category, we also include the prediction of potential application slowdown when running the application on a disaggregated memory system such as RAN.

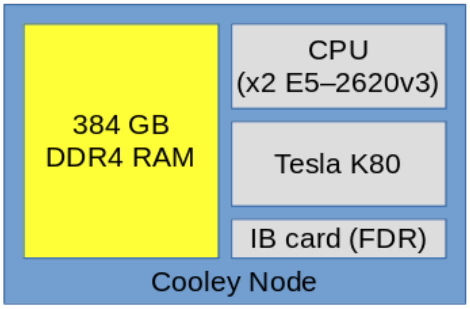

Fig. 1. The hardware in a Cooley node [18]

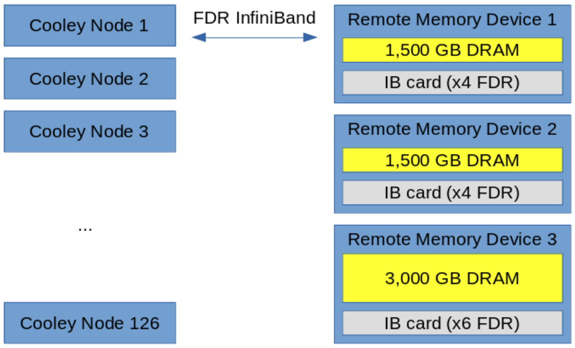

Fig. 2. The hardware in the RAN project [18]

Again, we'd like to emphasize how intentionally simple this model is. The idea behind this project is to develop a classification system that is accessible to any user with access to a machine that is able to run their benchmark. It's complicated enough to develop applications, especially for an emergent technology - we don't want to add undue complexity with our model, as that's contrary to the goal of demystifying application suitability on disaggregated memory.

A. Measuring Major Page Faults

Modern processor design accommodates major page fault measurement almost universally. Performance monitoring packages, such as the built-in Linux tool *perf*, can collect a system's major page faults over time and output these results to a CSV or even standard out. From there, it's fairly trivial for a programmer to collect the data and make a simple plot showing major page faults over time. For example, a runtime script using perf is available at our project's GitHub repository [22].

We chose this measurement based on the following factors. If we truly view our disaggregated memory system as an

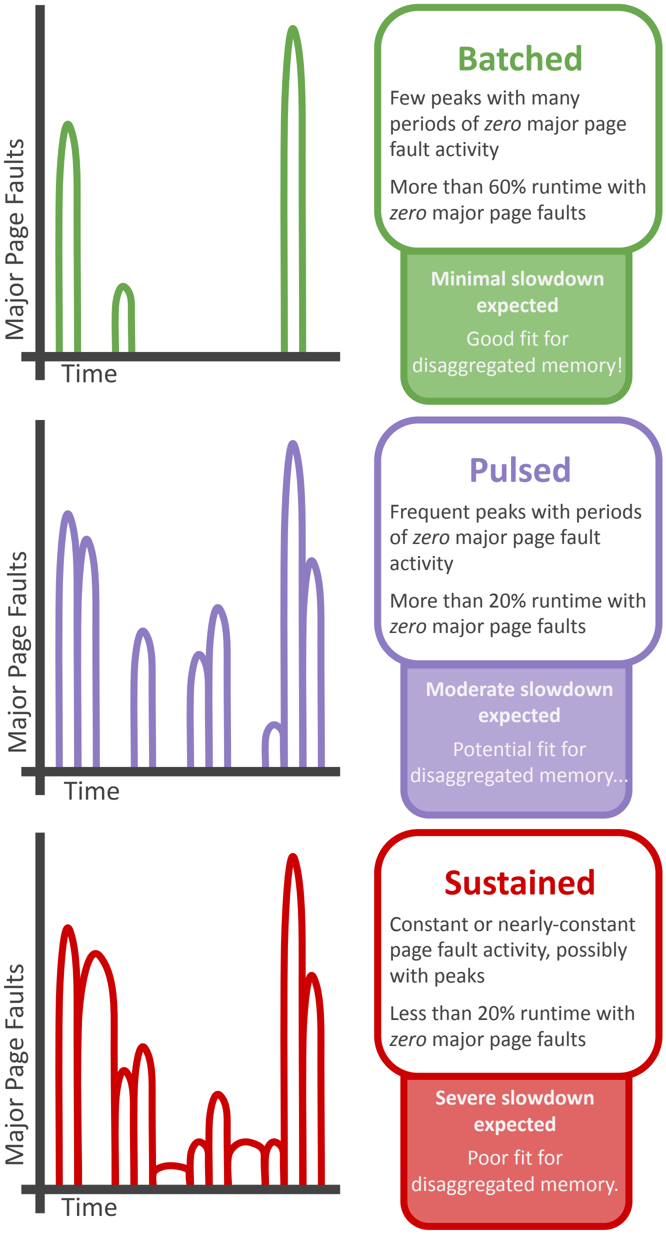
Major Page Faults
Time
Batched
Few peaks with many periods of *zero* major page fault activity
More than 60% runtime with *zero* major page faults
Minimal slowdown expected
Good fit for disaggregated memory!
Major Page Faults
Time
Pulsed
Frequent peaks with periods of *zero* major page fault activity
More than 20% runtime with *zero* major page faults
Moderate slowdown expected
Potential fit for disaggregated memory...
Major Page Faults
Time
Sustained
Constant or nearly-constant page fault activity, possibly with peaks
Less than 20% runtime with *zero* major page faults
Severe slowdown expected
Poor fit for disaggregated memory.

Fig. 3. Our classification model

extension to the typical memory hierarchy, then we can view the local memory as another (admittedly *much* slower) level to the cache and the remote memory occupies the hierarchical spot typically reserved for DRAM. Major page faults are thus equivalent to the last-level cache (LLC) miss rate, a well-documented and well-understood metric to determine if an application will work with the system's cache rather than against it [27]. Thus, we can use major page faults over time as a simple litmus test to determine if an application is well or poorly suited for running on a disaggregated memory system.

There are several potential pitfalls inherent to this measurement. First, the user must ensure they run their application on a system which *can* exhibit major page faults. The system must have storage - or something of equivalent use which the kernel can page out to.

Second, the user must ensure the application under study uses sufficient memory to *require* at least one page fault. Any application which uses more than a page of memory can exhibit a major page fault behavior, and while page sizes are configurable, the typical modern size of a Linux OS page is 4 MB [28]. To oversimplify, an application which does not use at least a page's worth of memory is unlikely to experience paging. Furthermore, if an application were very low in memory demand, it's unlikely a disaggregated memory architecture is needed at all.

Third, and this may go without saying, it's important the user runs an application which actually can page out to memory. Again, it's unlikely a user will want to make use of disaggregated memory without needing lots of memory, but it's good to be explicit at this stage. Additionally, it is recommended the user test their application on a system which can handle the computational load of the program but which also has insufficient local memory to complete the program's runtime fully in-memory. This way, the user can ensure they observe their application's major page fault behavior.

B. *Classification and Prediction*

Once a user collects their application's major page fault behavior, our model can provide classification and performance predictions.

We separate applications into three categories based on the measured major page faults over time: *batched*, *pulsed*, and *sustained*. There are simple delineations between these. A *batched* application will have few peaks with periods of zero major page fault activity separating the peaks, with at least 60% of the application's runtime displaying zero major page faults. A *pulsed* application will have frequent, possibly periodic peaks with periods of zero major page fault activity between the peaks, and it will spend between 60% and 20% of the runtime with zero measured major page faults. Finally, a *sustained* application will have constant or near-constant major page fault activity, possibly with peaks, but without periods of zero major page fault activity. It will spend less than 20% of its runtime with zero major page faults measured.

Any application falling into the *batched* category is expected to experience minimal performance slowdown if using disaggregated memory. An application fitting into the *pulsed* category can expect moderate slowdown, up to a 100% slowdown. And applications in the *sustained* category are expected to experience severe slowdown, well over 100%. Applications which exhibit *pulsed* behavior may do well on a disaggregated memory architecture with minor modifications to code structure, so if it is valuable to move to the novel system, it may be worth the user's time to attempt such modifications. And an application falling into the *batched* category is likely to run well on a disaggregated memory system out-of-the-box. See Fig. 3 for a visual representation of this model.

The delineations between categories were chosen based on extensive observation of a suite of representative applications from deep learning and scientific computing fields. Fig. 4 demonstrates the major page fault behaviors and categories of three applications which will be described in Section IV.

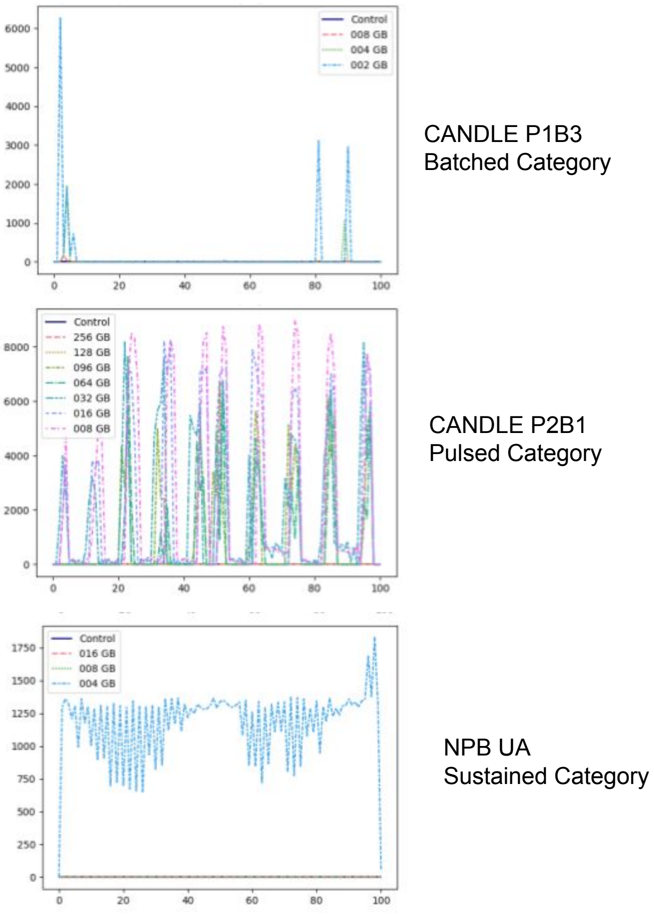

Fig. 4. Example runtime measurement and resulting classification from three benchmarks, x-axis represents time and y-axis measured major page faults per second

IV. Model Validation

In this section, we use our categorization model to examine nine benchmarks, predict their performance slowdown in a disaggregated memory environment, and verify the prediction against the actual results collected from running these benchmarks on the RAN testbed.

A. Benchmarks

We selected benchmarks from both the CANDLE and the NPB suites. The benchmarks were selected as they exhibit different runtime and memory behaviors, and the major task from each benchmark is given in Table I.

1) CANDLE: The CANcer Deep Learning Environment, or CANDLE, is an NIH-led Exascale Computing Project (ECP). Its goal is to build a scalable deep neural network (DNN) capable of simultaneously addressing three major concerns for cancer research (the “RAS pathway problem”, the “drug response problem”, and the “treatment strategy problem”). Each area of concern is at a different scale (cellular-level, molecular-level, and population-level), and each problem space has unique challenges [6].

TABLE I
Benchmark tasks

Benchmark	**Task**
CANDLE P1B1	Autoencoder Compressed Representation for Gene Expression
CANDLE P1B2	Sparse Classifier Disease Type Prediction from Somatic SNPs
CANDLE P1B3	MLP Regression Drug Response Prediction
CANDLE P2B1	Autoencoder Compressed Representation for Molecular Dynamics Simulation Data
CANDLE P3B1	Multi-task Deep Neural Net (DNN) for data extraction from clinical reports
NPB CG	Conjugate Gradient, smallest eigenvalue of sparse, symmetric matrix
NPB FT	Fast Fourier Transform, 3D partial differential equation solution
NPB IS	Integer Sort, typical sort method
NPB UA	Unstructured Adaptive mesh, solve stylized heat transfer problem

CANDLE has benchmarks separated over the three problem spaces. Pilot1 (P1) benchmarks concern problems and data at the cellular level, and the goal is to predict drug responses based on molecular data from tumors and drugs. Pilot2 (P2) benchmarks concern the molecular scale, and the goal is to assist in simulating molecular dynamics. Pilot3 (P3) benchmarks concern population-level data, and the goal is to predict cancer reappearance. The benchmark names follow the convention P#B#, where P# is either P1, P2, or P3 for the three different problem spaces, and B# is the benchmark number under that space. Our experiments cover benchmarks P1B1, P1B2, P1B3, P2B1, and P3B1.

2) NPB: The NASA Parallel Benchmark suite (NPB) is a set of benchmarks that measure performance on highly parallel systems. They are designed with versatility in mind, allowing users to vary the problem size as needed, while covering several fundamental algorithms [25]. We chose 4 benchmarks representing the cases of using irregular, strided, and random memory access as well as long-distance communication.

B. Benchmark Categorization, Prediction, and Observation

We first categorize each benchmark. Due to the page limit, we will not show graphs of major page faults over time for each benchmark, but such graphs are readily available in our github repository [22]. We then use our model to predict the magnitude of slowdown experienced by each benchmark, and we validate this against the measured slowdown when running on RAN.

It’s useful to discuss our experimental setup on RAN at this stage. ANL’s Cooley has nodes with 384 GB DRAM each. Each experiment has access to 100GB of DRAM located on the remote memory device. After each experiment terminates, the remote memory is wiped and returned to the available remote memory pool (see the background section on RAN as well as [5] for details). We run each experiment first using only local RAM, and given the size of the node, each benchmark runs in memory in this mode. We then run each benchmark with approximately 50% the needed memory available as a local memory cache; in other words, 50% of the data needed at any moment can fit in the local memory, but the remote memory will be required to complete the benchmark run. We then calculate the performance slowdown, defined in Formula (1), and see if this matches with our model's predictions. Here, S is the slowdown, $t_{control}$ is the application's runtime using local memory, and t is the application's runtime using remote memory.

$$S = \frac{t - t_{control}}{t_{control}} * 100\% \tag{1}$$

The benchmarks, their category under our model, their expected slowdown, and their observed slowdown is presented in Table II and graphically depicted in 5.

TABLE II
BENCHMARK CATEGORIES, PREDICTIONS, AND OBSERVATIONS

Benchmark	**Category**	**Expected Slowdown**	**Average Slowdown**
CANDLE P1B1	Batched	Minimal	9%
CANDLE P1B2	Pulsed	Moderate	27%
CANDLE P1B3	Batched	Minimal	-22%
CANDLE P2B1	Pulsed	Moderate	100%
CANDLE P3B1	Batched	Minimal	-15%
NPB CG	Sustained	Severe	291%
NPB FT	Sustained	Severe	923%
NPB IS	Sustained	Severe	421%
NPB UA	Sustained	Severe	305%

We thus observed the expected slowdown for all benchmarks under our categorization model. Of note, some benchmarks appear to perform *faster* when using disaggregated memory. Investigating this suggests driver differences are the cause; the proprietary drivers in the RAN system use a different version of malloc that is more efficient in some uses.

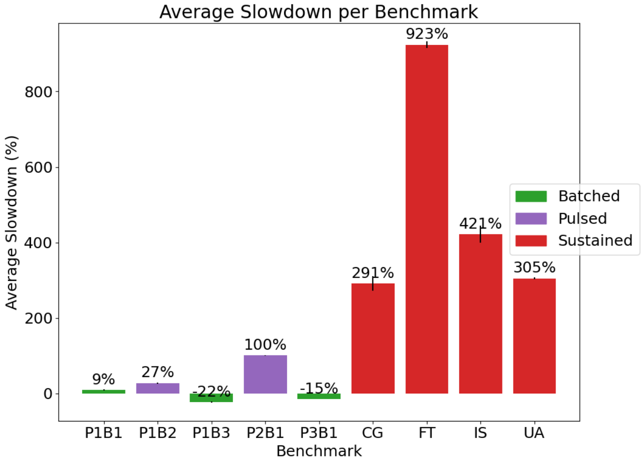

Fig. 5. Correlation between model classification and performance slowdown

V. CONCLUSIONS

The above experimental study demonstrates the effectiveness of our categorization model and provides a way to answer to the question raised in Section I, that is, how will an application perform on a disaggregated memory architecture?

We propose a simple, replicatable, reliable method for estimating an application's performance when using disaggreagated memory. An application exhibiting *sustained* major page fault behavior across varied local RAM sizes experiences a steep increase in performance slowdown the more remote memory is used. In contrast, applications exhibiting *pulsed* or *batched* major page fault behavior experience little to no performance slowdown regardless of how intensely the remote memory is used, and thus, they are suitable candidates for running on a disaggregated memory architecture. It is straightforward to characterize applications using their major page fault behaviors (using standard hardware and libraries), and the resulting category indicates whether or not an application is a good fit for using disaggregated memory.

This study also demonstrates the benefits of a disaggregated memory architecture for a range of applications. It is entirely possible to reduce node requirements without severely impacting the performance of a workload as long as that workload falls into the pulsed or infrequent major page fault patterns. For instance, for the memory-intensive benchmark (P2B1), using a local RAM cache as little as 64 GB (a local memory reduction of over 80%!) , we observe a slowdown of 30%. This shows, using remote memory, per-node memory can be reduced to meet an average workload rather than requiring some or all nodes to accommodate the most demanding workloads. Given the demand and expense of memory, this is certainly a positive outlook and deserves increased study.

Our work is intended to offer empirically-backed guidance for running applications on a disaggregated memory system; significant further work is needed to generalize our findings concretely and broadly. To that end, our final suggested use of this work is as a spring-board into further empirical study and modeling of application performance when using disaggregated memory.

As HPC systems become increasingly heterogeneous, it is becoming increasingly difficult to predict ahead of time what applications are a suitable fit for a new architecture. We hope this study formalizes and demystifies predicting application performance on disaggregated memory.

ACKNOWLEDGMENTS

This work is supported in part by US National Science Foundation grants CCF-1618776 and CNS-1717763.

This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

REFERENCES

[1] G. Zervas et al., ”Disaggregated compute, memory and network systems: A new era for optical data centre architectures,” 2017 Optical Fiber Communications Conference and Exhibition (OFC), Los Angeles, CA, 2017, pp. 1-3.

[2] Zeshan Chishti and Berkin Akin. 2019. Memory system characterization of deep learning workloads. In Proceedings of the International Symposium on Memory Systems (MEMSYS '19). Association for Computing Machinery, New York, NY, USA, 497–505.

[3] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. 2009. Disaggregated memory for expansion and sharing in blade servers. In Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09). Association for Computing Machinery, New York, NY, USA, 267–278.

[4] Dhantu Buragohain, Abhishek Ghogare, Trishal Patel, Mythili Vutukuru, and Purushottam Kulkarni. 2017. DiME: A Performance Emulator for Disaggregated Memory Architectures. In Proceedings of the 8th Asia-Pacific Workshop on Systems (APSys '17). Association for Computing Machinery, New York, NY, USA, Article 15, 1–8.

[5] Dawid Zawislak, Brian Toonen, William Allcock, Silvio Rizzi, Joseph Insley, Venkatram Vishwanath, and Michael E. Papka. 2016. Early investigations into using a remote ram pool with the vl3 visualization framework. In Proceedings of the 2nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization (ISAV '16). IEEE Press, 23–28.

[6] Department of Energy, CANcer Distributed Learning Environment [Source code repository]. https://github.com/ECP-CANDLE/Candle

[7] J. Li, "In-memory computing for scalable data analytics," 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, 2015, pp. 93-94.

[8] M. Bielski et al., "dReDBox: Materializing a full-stack rack-scale system prototype of a next-generation disaggregated datacenter," 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, 2018, pp. 1093-1098.

[9] Vamsee Reddy Kommareddy, Amro Awad, Clayton Hughes, and Simon David Hammond. 2018. Exploring Allocation Policies in Disaggregated Non-Volatile Memories. In Proceedings of the Workshop on Memory Centric High Performance Computing (MCHPC'18). Association for Computing Machinery, New York, NY, USA, 58–66.

[10] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker. Network requirements for resource disaggregation. In Proc. of the OSDI'16.

[11] W. Allcock et al., "RAM as a Network Managed Resource," 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, 2018, pp. 99-106.

[12] A. D. Papaioannou, R. Nejabati and D. Simeonidou, "The Benefits of a Disaggregated Data Centre: A Resource Allocation Approach," 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, 2016, pp. 1-7.

[13] Anderson, Eric Arvid and Jeanna M. Neefe. "An Exploration of Network RAM." (1998).

[14] Pramod Subba Rao and George Porter. 2016. Is Memory Disaggregation Feasible? A Case Study with Spark SQL. In Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems (ANCS '16). Association for Computing Machinery, New York, NY, USA, 75–80.

[15] Josue V. Quiroga, Marti Torrents, Nehir Sonmez, Dimitris Theodoropoulos, Ferad Zyulkyarov, and Mario Nemirovsky. 2019. Evaluation of a Rack-Scale Disaggregated Memory Prototype for Cloud Data Centers. In Proceedings of the 30th International Workshop on Rapid System Prototyping (RSP'19) (RSP '19). Association for Computing Machinery, New York, NY, USA, 15–21.

[16] Y. Kwon and M. Rhu, "A Disaggregated Memory System for Deep Learning," in IEEE Micro, vol. 39, no. 5, pp. 82-90, 1 Sept.-Oct. 2019.

[17] Allcock, William E. "RAN – RAM Area Network" Argonne National Laboratory Joint Laboratory for System Evaluation. https://press3.mcs.anl.gov/jlse/projects/ran-ram-area-network/ (accessed Apr. 21, 2020)

[18] "Cooley System Overview" Argonne Leadership Computing Facility. https://www.alcf.anl.gov/support-center/cooley/cooley-system-overview (accessed Apr. 21, 2020)

[19] Mellanox Technologies. "Introduction to InfiniBand" https://www.mellanox.com/pdf/whitepapers/IB_Intro_WP_190.pdf (accessed Apr. 21, 2020)

[20] J. Chae, D. Bhowmik, H. Ma, A. Ramanathan and C. Steed, "Visual Analytics for Deep Embeddings of Large Scale Molecular Dynamics Simulations," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 1759-1764.

[21] Department of Energy, CANcer Distributed Learning Environment Benchmarks [source code commit] https://github.com/ECP-CANDLE/Benchmarks/pull/57/commits/cdd9c478ad83f6fd8e07f9 e0f2504d6ea612c337

[22] Cornelius, Melanie and Peck, Avery, Public CANDLE results repository [source code repository] https://github.com/mseryn/CANDLE_Testing_public

[23] J. Taylor Childers, Thomas D. Uram, Doug Benjamin, Thomas J. LeCompte, and Michael E. Papka. 2017. An Edge Service for Managing HPC Workflows. In Proceedings of the Fourth International Workshop on HPC User Support Tools (HUST'17). Association for Computing Machinery, New York, NY, USA, Article 1, 1–8.

[24] K. Awedat, M. Alajmi and J. R. Springstead, "Compressive sensing of jointly sparse signals as a method for dimensionality reduction of mass spectrometry data," 2017 IEEE International Conference on Electro Information Technology (EIT), Lincoln, NE, 2017, pp. 080-085.

[25] "NAS Parallel Benchmarks" NASA Advanced Supercomputing Division. https://www.nas.nasa.gov/publications/npb.html

[26] "Memory Resource Controller" The Linux Kernel Archives. https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt

[27] Average Memory Access Time. John L. Hennessy and David A. Patterson, Computer Architecture a Quantitative Approach Fifth Edition, 2012, pp.B9-B19

[28] Using 4KB Page Size for Virtual Memory is Obsolete". IEEE. 2009-08-10.
Company, City, State, Country, Rep. no., (optional: vol./issue), Date. Accessed: Date. [Online]. Available: site/path/file