Architectural Support for Efficient Data Movement in Disaggregated Systems

I. Data Movement in Disaggregated Systems

Traditional data centers include monolithic servers that tightly integrate CPU, memory and disk (Fig. 1a). Instead, *Disaggregated Systems* (**DSs**) [1]–[3] organize multiple compute (**CC**), memory (**MC**) and storage devices as *independent, failure-isolated* components interconnected over a high-bandwidth network (Fig. 1b). DSs can greatly reduce data center costs by providing improved resource utilization, resource scaling, failure-handling and elasticity in modern data centers [1]–[4].
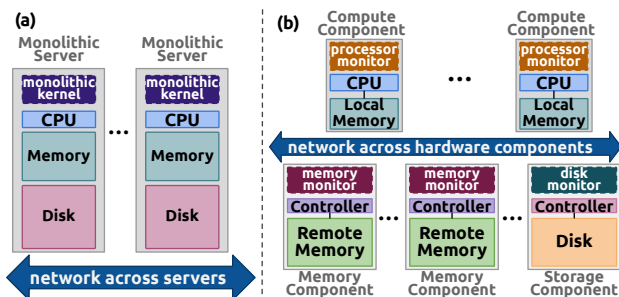

Fig. 1. (a) Traditional systems vs (b) DSs.

The MCs provide large pools of main memory (***remote memory***), while the CCs include the on-chip caches and a few GBs of DRAM (***local memory***) that acts as a cache of *remote memory*. In this context, a large fraction of the application's data ($\sim 80\%$) [1], [3], [5] is located in *remote memory*, and can cause large performance penalties (see Fig. 2) from remotely accessing data over the network.

Alleviating data access overheads is challenging in DSs for the following reasons. First, DSs are not monolithic and comprise independently managed entities: each component has its own hardware controller, and a specialized kernel monitor uses its own functionality to manage the component it runs on (only communicates with other monitors via network messaging if there is a need to access remote resources). This characteristic necessitates a distributed and disaggregated solution that can scale to a large number of independent components in the system. Second, there is high variability in remote memory access latencies since they depend on the locations of the MCs, contention from other jobs that share the same network and MCs, and data placements that can vary during runtime or between multiple executions. This necessitates a solution that is robust towards fluctuations in the network/remote memory bandwidth and latencies. Third, a major factor behind the performance slowdowns is the commonly-used approach in DSs [1], [3]–[5] of moving data at page granularity. This approach effectively provides software transparency, low metadata costs in memory management, and high spatial locality in many applications. However, it can cause high bandwidth consumption and network congestion,

and often significantly slows down accesses to critical path cache lines in other concurrently accessed pages.

Fig. 2 presents a performance analysis of different data movement schemes using DSs for a representative set of heterogeneous workloads. We evaluate a MC and a CC with *local memory* to fit $\sim 20\%$ of the application data, and the network bandwidth to be $1/2$-$1/8\times$ of the bus bandwidth [1], [5]. Performance of all schemes is normalized to that of the monolithic approach where *all* data fits in the *local memory* of the CC. We make three observations.
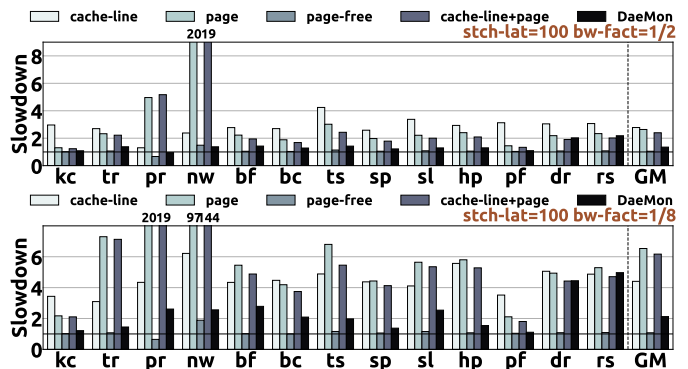

Fig. 2. Data movement overheads in DSs.

First, the typically-used *page* scheme of moving data at a page granularity incurs large slowdowns over the monolithic *Local* configuration due to transferring large amounts of data over the network that slows down accesses to other pages. Instead, when pages are moved *for free*, i.e., *page-free*, performance significantly improves thanks to spatial locality benefits within pages. Second, moving data always at a *fixed* granularity (*cache-line* via LLC or *page* via *local memory*) cannot provide robustness across heterogeneous applications and network configurations: some applications can benefit from cache line (e.g., *pr*, *nw*) or page granularity (e.g., *pf*, *dr*) data movements, while the best-performing granularity also depends on network characteristics (e.g., *bf*, *ts*). Third, *naively* moving data at *both* granularities (*cache-line+page*) to serve data requests with the latency of the packet that arrives earlier to the CC is still quite inefficient, while critical cache line requests are still queued behind large concurrently accessed pages. In contrast, Fig. 2 shows that *DaeMon* (see §III) significantly reduces data movement overheads in DSs across various network/application characteristics.

II. Prior Work

Prior works [1]–[25] propose OS kernels, system-level solutions, software management systems, architectures for DSs. These works do not tackle the data movement challenge in DSs, and thus our work is orthogonal to them.

Prior works on hybrid systems [26]–[38] integrate die-stacked DRAM [39] as DRAM cache of a large main memory [26], [28], [31] in a monolithic server, and tackle high page

movement costs in two-tiered physical memory via page placement/hot page selection schemes or by moving data at smaller granularity, e.g., cache line. However, data movement in DSs poses fundamentally different challenges. First, accesses across the network are significantly slower than within the server, thus intelligent page placement/movement cannot by itself address these high costs. Second, DSs incur significant variations in access latencies based on the current network architecture and concurrent jobs sharing the MCs/network, thus necessitating an solution primarily designed for robustness to this variability. Finally, DSs include independently managed MCs and networks shared by independent CCs running unknown jobs. Thus, unlike hybrid systems, the solution cannot assume that the memory management at the MCs can be fully controlled by the CPU side. Our work is the first to examine the data movement problem in fully DSs, and design an effective solution for DSs.

III. *DaeMon*'s KEY IDEAS

DaeMon (Fig. 3) is an adaptive and scalable mechanism to alleviate data costs in DSs, consisting of three techniques.
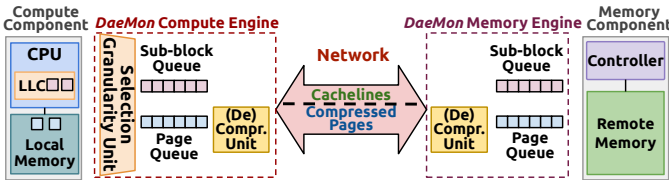

Fig. 3. High-level overview of *DaeMon*.

(1) Decoupled Multiple Granularity Data Movement. We integrate two separate hardware queues to serve data requests from *remote memory* at two granularities, i.e., cache line (via the *sub-block queue* to LLC) and page (via the *page queue* to *local memory*) granularity, and effectively *prioritize* moving cache lines over moving pages via a *bandwidth partitioning* approach: a queue controller serves cache line requests with a higher *predefined fixed rate* than page requests to ensure that any given time a certain fraction of the bandwidth resources is always allocated to serve cache line moves fast. This key technique enables (i) low metadata overheads by retaining page migrations, (ii) high performance by leveraging data locality within pages, and (iii) fewer slowdowns in cache line data movements that are on the critical path, from expensive page moves that may have been previously triggered.

(2) Selection Granularity Data Movement. To provide an adaptive data movement solution, we include in each CC two separate hardware buffers to track pending data migrations for both cache line (via the *inflight sub-block buffer*) and page (via the *inflight page buffer*) granularity, and a selection granularity unit to decide if a data request should be served by cache line, page or *both* based on the utilization of the *inflight* buffers. Given that *DaeMon* prioritizes cache line over page moves, the inflight buffers are utilized in different ways, allowing us to capture the application behavior and the system load during runtime. For example: (a) If there is *low locality* within pages, the page buffer has higher utilization than the sub-block buffer (cache lines are prioritized), thus the selection

unit favors moving cache lines and throttles pages (and vice-versa). (b) Under low *bandwidth utilization* scenarios, the page buffer utilization is low, thus the selection unit schedules more page movements (or both granularities) to obtain data locality benefits (and vice-versa).

(3) Link Compression on Page Movements. We employ hardware compression units at both the CCs and MCs to highly compress pages migrated over the network. *Link compression* on page moves reduces the network bandwidth consumption and alleviates network bottlenecks.

Synergy of three techniques. *DaeMon* cooperatively integrates all three techniques to significantly alleviate data movement overheads, and provide robustness towards network, architecture and application characteristics:

(1) Prioritizing requested cache lines helps *DaeMon* to tolerate high (de)compression latencies in page migrations over the network, while also leveraging benefits of page migrations (low metadata costs, spatial locality in pages).

(2) Compressed pages consume less network bandwidth, enabling *DaeMon* to reserve part of the bandwidth to effectively prioritize critical path cache line accesses.

(3) Compression on page moves helps *DaeMon* to adapt to the data compressibility: if the pages are highly compressible, the inflight page buffer empties at a faster rate, and *DaeMon* favors sending more pages (and vice-versa).

IV. *DaeMon*'s KEY RESULTS

We extend Sniper [40] to simulate DSs, and evaluate capacity intensive workloads from graph processing, HPC, data analytics, bioinformatics, machine learning domains. *DaeMon* reduces data access costs and improves performance by 3.06× and 2.39×, respectively, over the widely-adopted approach of moving data at a page granularity. *DaeMon* leverages the synergy of all three techniques to provide robustness, while retaining the spatial locality, transparency and metadata management benefits of page granularity movements. We show that *DaeMon* achieves high system performance for various network/architecture configurations and applications (Fig. 4 top), and multiple concurrently running jobs in the DS (Fig. 4 bottom), compared to the widely-adopted approach of moving data at page granularity.
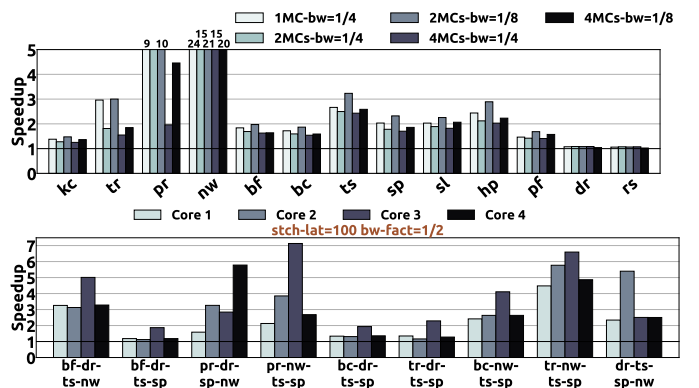

Fig. 4. *DaeMon*'s benefits over the *page* scheme, (i) varying the MCs, network bandwidth and application, and (ii) running multiple applications in a 4-CPU CC and a MC.

REFERENCES

[1] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. Legoos: A disseminated, distributed os for hardware resource disaggregation. In *OSDI*, OSDI'18, page 69–87, USA, 2018. USENIX Association.

[2] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. Clio: A hardware-software co-designed disaggregated memory system, 2022.

[3] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. Mind: In-network memory management for disaggregated data centers. In *SOSP*, 2021.

[4] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *ASPLOS*, ASPLOS 2021, page 79–92, 2021.

[5] Peter X. Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *OSDI*, OSDI'16, page 249–264, 2016.

[6] Sebastian Angel, Mihir Nanavati, and Siddhartha Sen. Disaggregation and the application. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, July 2020.

[7] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: Practical and energy-efficient memory disaggregation in a datacenter. In *EuroSys*, 2018.

[8] Sangjin Han, Norbert Egi, Aurojit Panda, Sylvia Ratnasamy, Guangyu Shi, and Scott Shenker. Network support for resource disaggregation in next-generation datacenters. In *HotNets*, 2013.

[9] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote memory in the age of fast networks. In *SoCC*, SoCC '17, page 121–127, 2017.

[10] Qizhen Zhang, Yifan Cai, Sebastian G. Angel, Vincent Liu, Ang Chen, and B. T. Loo. Rethinking data management systems for disaggregated data centers. In *CIDR*, 2020.

[11] Kevin Lim, Yoshio Turner, Jose Renato Santos, Alvin AuYoung, Jichuan Chang, Parthasarathy Ranganathan, and Thomas F. Wenisch. System-level implications of disaggregated memory. In *HPCA*, 2012.

[12] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A memory-disaggregated managed runtime. In *OSDI*, pages 261–280. USENIX Association, November 2020.

[13] Pengfei Zuo, Jiazhao Sun, Liu Yang, Shuangwu Zhang, and Yu Hua. One-sided rdma-conscious extendible hashing for disaggregated memory. In *USENIX ATC*, pages 15–29. USENIX Association, July 2021.

[14] Laurent Bindschaedler, Ashvin Goel, and Willy Zwaenepoel. Hailstorm: Disaggregated compute and storage for distributed lsm-based databases. In *ASPLOS*, ASPLOS '20, page 301–316, 2020.

[15] Ivy Peng, Roger Pearce, and Maya Gokhale. On the memory underutilization: Exploring disaggregated memory on hpc systems. In *SBAC-PAD*, 2020.

[16] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. Software-defined far memory in warehouse-scale computers. In *ASPLOS*, ASPLOS '19, page 317–330, 2019.

[17] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *NSDI*, 2017.

[18] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novaković, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote regions: a simple abstraction for remote memory. In *USENIX ATC*, 2018.

[19] Christian Pinto, Dimitris Syrivelis, Michele Gazzetti, Panos Koutsovasilis, Andrea Reale, Kostas Katrinis, and H. Peter Hofstee. Thymesisflow: A software-defined, hw/sw co-designed interconnect stack for rack-scale memory disaggregation. In *MICRO*, pages 868–880, 2020.

[20] K. Katrinis, D. Syrivelis, D. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. Lopez-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends. Rack-scale disaggregated cloud data centers: The dredbox project vision. In *DATE*, 2016.

[21] Dhantu Buragohain, Abhishek Ghogare, Trishal Patel, Mythili Vutukuru, and Purushottam Kulkarni. Dime: A performance emulator for disaggregated memory architectures. In *APSys*, APSys '17, 2017.

[22] Pramod Subba Rao and George Porter. Is memory disaggregation feasible? a case study with spark sql. In *ANCS*, ANCS '16, page 75–80, 2016.

[23] Georgios Zervas, Hui Yuan, Arsalan Saljoghei, Qianqiao Chen, and Vaibhawa Mishra. Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation [invited]. *JOCN*, 2018.

[24] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. Direct access, High-Performance memory disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022.

[25] Yang Zhou, Hassan M. G. Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David E. Culler, Henry M. Levy, and Amin Vahdat. Carbink: Fault-Tolerant far memory. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022.

[26] Xiangyu Dong, Yuan Xie, Naveen Muralimanohar, and Norman P. Jouppi. Simple but effective heterogeneous main memory with on-chip memory controller support. In *SC*, 2010.

[27] Haikun Liu, Yujie Chen, Xiaofei Liao, Hai Jin, Bingsheng He, Long Zheng, and Rentong Guo. Hardware/software cooperative caching for hybrid dram/nvm memory architectures. In *ICS*, ICS '17, 2017.

[28] Xiaowei Jiang, Niti Madan, Li Zhao, Mike Upton, Ravishankar Iyer, Srihari Makineni, Donald Newell, Yan Solihin, and Rajeev Balasubramonian. Chop: Adaptive filter-based dram caching for cmp server platforms. In *HPCA*, 2010.

[29] Jagadish B. Kotra, Haibo Zhang, Alaa R. Alameldeen, Chris Wilkerson, and Mahmut T. Kandemir. Chameleon: A dynamically reconfigurable heterogeneous memory system. In *MICRO*, 2018.

[30] Andreas Prodromou, Mitesh Meswani, Nuwan Jayasena, Gabriel Loh, and Dean M. Tullsen. Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories. In *HPCA*, 2017.

[31] Neha Agarwal and Thomas F. Wenisch. Thermostat: Application-transparent page management for two-tiered main memory. In *ASPLOS*, ASPLOS '17, 2017.

[32] Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In *HPCA*, 2015.

[33] Kai Wu, Yingchao Huang, and Dong Li. Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory. In *SC*, 2017.

[34] Gabriel H. Loh and Mark D. Hill. Efficiently enabling conventional block sizes for very large die-stacked dram caches. In *MICRO*, 2011.

[35] Gabriel Loh and Mark D. Hill. Supporting very large dram caches with compound-access scheduling and missmap. *IEEE Micro*, 2012.

[36] Chia Chen Chou, Aamer Jaleel, and Moinuddin K. Qureshi. Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *MICRO*, 2014.

[37] Jee Ho Ryoo, Mitesh R. Meswani, Andreas Prodromou, and Lizy K. John. Silc-fm: Subblocked interleaved cache-like flat memory organization. In *HPCA*, 2017.

[38] Djordje Jevdjic, Stavros Volos, and Babak Falsafi. Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache. In *ISCA*, ISCA '13, 2013.

[39] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. HBM DRAM Technology and Architecture. In *IMW*, 2017.

[40] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *SC*, pages 52:1–52:12, November 2011.